

Deep Latent Variable Models of Natural Language

A DISSERTATION PRESENTED
BY
YOON H. KIM
TO
THE DEPARTMENT OF SCHOOL OF ENGINEERING AND APPLIED SCIENCES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE SUBJECT OF
COMPUTER SCIENCE

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS
MAY 2020

©2020 – YOON H. KIM
ALL RIGHTS RESERVED.

Deep Latent Variable Models of Natural Language

ABSTRACT

Understanding natural language involves complex underlying processes by which meaning is extracted from surface form. One approach to operationalizing such phenomena in computational models of natural language is through probabilistic latent variable models, which can encode structural dependencies among observed and unobserved variables of interest within a probabilistic framework. Deep learning, on the other hand, offers an alternative computational approach to modeling natural language through end-to-end learning of expressive, global models, where any phenomena necessary for the task are captured implicitly within the hidden layers of a neural network. This thesis explores a synthesis of deep learning and latent variable modeling for natural language processing applications. We study a class of models called *deep latent variable models*, which parameterize components of probabilistic latent variable models with neural networks, thereby retaining the modularity of latent variable models while at the same time exploiting rich parameterizations enabled by recent advances in deep learning. We experiment with different families of deep latent variable models to target a wide range of language phenomena—from word alignment to parse trees—and apply them to core natural language processing tasks including language modeling, machine translation, and unsupervised parsing.

We also investigate key challenges in learning and inference that arise when working with deep latent variable models for language applications. A standard approach

for learning such models is through amortized variational inference, in which a global inference network is trained to perform approximate posterior inference over the latent variables. However, a straightforward application of amortized variational inference is often insufficient for many applications of interest, and we consider several extensions to the standard approach that lead to improved learning and inference. In summary, each chapter presents a deep latent variable model tailored for modeling a particular aspect of language, and develops an extension of amortized variational inference for addressing the particular challenges brought on by the latent variable model being considered. We anticipate that these techniques will be broadly applicable to other domains of interest.

Contents

1	INTRODUCTION	1
1.1	Thesis Outline	3
1.2	Related Publications	4
2	BACKGROUND	6
2.1	Motivation	6
2.2	Notation	9
2.3	Neural Networks	11
2.4	Latent Variable Models	13
2.4.1	Deep Latent Variable Models	15
2.5	Learning and Inference	18
2.5.1	Maximum Likelihood Estimation	19
2.5.2	Tractable Exact Inference	20
2.5.3	Variational Inference	23
2.5.4	Amortized Variational Inference	28
2.5.5	Variational Autoencoders	31
2.6	Thesis Roadmap	37
3	LATENT VARIABLE MODEL OF SENTENCES & SEMI-AMORTIZED VARIATIONAL INFERENCE	38
3.1	Introduction	38
3.2	Background	39
3.2.1	Generative Model	40
3.2.2	Posterior Collapse	42
3.2.3	Amortization Gap	44
3.3	Semi-Amortized Variational Autoencoders	45
3.3.1	Implementation Details	47
3.4	Empirical Study	49
3.4.1	Experimental Setup	49

3.4.2	Results	51
3.4.3	Analysis of Latent Variables	54
3.5	Discussion	57
3.5.1	Limitations	57
3.5.2	Posterior Collapse: Optimization vs. Underlying Model	58
3.6	Related Work	59
3.7	Conclusion	60
4	LATENT VARIABLE MODEL OF ATTENTION & RELAXATIONS OF DISCRETE SPACES	61
4.1	Introduction	61
4.2	Background	63
4.2.1	Soft Attention	64
4.2.2	Hard Attention	66
4.2.3	Pathwise Gradient Estimators for Discrete Distributions	68
4.3	Variational Attention for Latent Alignment	71
4.3.1	Test Time Inference	76
4.4	Empirical Study	77
4.4.1	Experimental Setup	77
4.4.2	Results	79
4.4.3	Analysis	81
4.5	Discussion	84
4.5.1	Limitations	84
4.5.2	Attention as a Latent Variable	84
4.6	Related Work	86
4.7	Conclusion	87
5	LATENT VARIABLE MODEL OF TREES & POSTERIOR REGULARIZATION	88
5.1	Introduction	88
5.2	Background	90
5.2.1	Recurrent Neural Network Grammars	90
5.2.2	Posterior Regularization	91
5.2.3	Conditional Random Fields	93
5.3	Unsupervised Recurrent Neural Network Grammars	94
5.3.1	Generative Model	94
5.3.2	Posterior Regularization with Conditional Random Fields . . .	97
5.3.3	Learning and Inference	101
5.4	Empirical Study	104
5.4.1	Experimental Setup	104

5.4.2	Results	108
5.4.3	Analysis	113
5.5	Discussion	115
5.5.1	Limitations	115
5.5.2	Rich Generative Models for Learning Latent Structures	116
5.6	Related Work	117
5.7	Conclusion	118
6	LATENT VARIABLE MODEL OF GRAMMARS & COLLAPSED VARIATIONAL INFERENCE	120
6.1	Introduction	120
6.2	Background	123
6.2.1	Probabilistic Context-Free Grammars	123
6.2.2	Grammar Induction vs. Unsupervised Parsing	125
6.3	Compound Probabilistic Context-Free Grammars	125
6.3.1	A Neural Parameterization	126
6.3.2	A Compound Extension	128
6.3.3	Training and Inference	131
6.4	Empirical Study	134
6.4.1	Experimental Setup	134
6.4.2	Results	136
6.4.3	Analysis	138
6.4.4	Recurrent Neural Network Grammars on Induced Trees	142
6.5	Discussion	145
6.5.1	Limitations	145
6.5.2	Richer Grammars for Modeling Natural Language	146
6.6	Related Work	147
6.7	Conclusion	148
7	CONCLUSION	150
	REFERENCES	174

Listing of figures

2.1	Graphical model for the Naive Bayes model. For simplicity, all sequences are depicted as having T tokens. All distributions are categorical, and the parameters are $\boldsymbol{\mu} \in \Delta^{K-1}$ and $\boldsymbol{\pi} = \{\boldsymbol{\pi}_k \in \Delta^{V-1}\}_{k=1}^K$	15
2.2	Graphical model representation of a categorical latent variable model with tokens generated by an RNN. For simplicity, all sequences are depicted as having T tokens. The $z^{(n)}$'s are drawn from a Categorical distribution with parameter $\boldsymbol{\mu}$, while $x^{(n)}$ is drawn from an RNN. The parameters of the RNN are given by $\theta = \{\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{T}, \mathbf{E}, \mathbf{b}\}$. See the text for more details. . .	18
2.3	(Top) Traditional variational inference uses variational parameters $\lambda^{(n)}$ for each data point $x^{(n)}$. (Bottom) Amortized variational inference employs a global inference network ϕ that is run over the input $x^{(n)}$ to produce the local variational distributions.	29
3.1	(Left) Perplexity upper bound of various models when trained with 20 steps (except for VAE) and tested with varying number of SVI steps from random initialization. (Right) Same as the left except that SVI is initialized with variational parameters obtained from the inference network. . . .	53
3.2	(Top) Output saliency visualization of some examples from the test set. Here the saliency values are rescaled to be between 0-100 within each example for easier visualization. Red indicates higher saliency values. (Middle) Input saliency of the first test example from the top (in blue), in addition to two sample outputs generated from the variational posterior (with their saliency values in red). (Bottom) Same as the middle except we use a made-up example.. . . .	55
3.3	Saliency by part-of-speech tag, position, and log frequency for the output (top) and the input (bottom). See text for the definitions of input/output saliency. The dotted gray line in each plot shows the average saliency across all words.	57

4.1	Sketch of variational attention applied to machine translation. Here the source sentence $x_{1:7}$ has 7 words and the target sentence $y_{1:9}$ has 9 words. Two alignment distributions are shown, for the blue prior p , and the red variational posterior q taking into account future observations. Our aim is to use q , which conditions on the entire target sentence, to improve estimates of p and to support improved inference of z	73
4.2	Test perplexity of different approaches while varying K to estimate $\log p(y_t x, y_{<t}; \theta)$. Dotted lines compare soft baseline and variational with full enumeration.	81
4.3	(Left Column) Examples highlighting the difference between the prior attention $p(z_t x, y_{<t}; \theta)$ (red) and the variational posterior $q(z_t x, y; \phi)$ (blue) when translating from German to English (left-to-right). The variational posterior is able to better handle reordering; in (a) the variational posterior successfully aligns ‘turning’ to ‘verwandelt’, in (c) we see a similar pattern with the alignment of the clause ‘that’s my brand’ to ‘das ist meine marke’. (Right Column) Comparisons between soft attention (green) and the prior attention from a model trained with variational attention (red). Alignments from both models are similar, but variational attention is lower entropy. Both soft and variational attention rely on aligning the inserted English word ‘orientation’ to the comma in (b) since a direct translation does not appear in the German source.	83
5.1	Overview of our approach. The inference network $q(\mathbf{z} x; \phi)$ (left) is a CRF parser which produces a distribution over binary trees (shown in dotted box). \mathbf{B}_{ij} are random variables for existence of a constituent spanning i -th and j -th words, whose potentials are the output from a bidirectional LSTM (the global factor ensures that the distribution is only over valid binary trees). The generative model $p(x, \mathbf{z}; \theta)$ (right) is an RNNG which consists of a stack LSTM (from which actions/words are predicted) and a tree LSTM (to obtain constituent representations upon REDUCE). Training involves sampling a binary tree from $q(\mathbf{z} x; \phi)$, converting it to a sequence of shift/reduce actions ($\mathbf{z} = [\text{SHIFT}, \text{SHIFT}, \text{SHIFT}, \text{REDUCE}, \text{REDUCE}, \text{SHIFT}, \text{REDUCE}]$ in the above example), and optimizing the log joint likelihood $\log p(x, \mathbf{z}; \theta)$	102
5.2	Perplexity of the different models grouped by sentence length on PTB.	109

- 6.1 A graphical model-like diagram for the neural PCFG (left) and the compound PCFG (right) for an example tree structure. In the above, $A_1, A_2 \in \mathcal{N}$ are nonterminals, $T_1, T_2, T_3 \in \mathcal{P}$ are preterminals, $w_1, w_2, w_3 \in \Sigma$ are terminals. In the neural PCFG, the global rule probabilities $\boldsymbol{\pi} = \pi_S \cup \pi_{\mathcal{N}} \cup \pi_{\mathcal{P}}$ are the output from a neural net run over the symbol embeddings $\mathbf{E}_{\mathcal{G}}$, where $\pi_{\mathcal{N}}$ are the set of rules with a nonterminal on the left hand side (π_S and $\pi_{\mathcal{P}}$ are similarly defined). In the compound PCFG, we have per-sentence rule probabilities $\boldsymbol{\pi}_{\mathbf{z}} = \pi_{\mathbf{z},S} \cup \pi_{\mathbf{z},\mathcal{N}} \cup \pi_{\mathbf{z},\mathcal{P}}$ obtained from running a neural net over a random vector \mathbf{z} (which varies across sentences) and global symbol embeddings $\mathbf{E}_{\mathcal{G}}$. In this case, the context-free assumptions hold conditioned on \mathbf{z} , but they do not hold unconditionally: e.g. when conditioned on \mathbf{z} and A_2 , the variables A_1 and T_1 are independent; however when conditioned on just A_2 , they are not independent due to the dependence path through \mathbf{z} . Note that the rule probabilities are random variables in the compound PCFG but deterministic variables in the neural PCFG.129

Acknowledgments

First and foremost I would like to thank my advisor, Sasha Rush. I have benefited immeasurably from his guidance throughout the Ph.D., from technical advice on how to best implement semiring algebra on GPUs to discussions on how to write and present papers, and everything in between.

I am also grateful to my committee members Finale Doshi-Velez and Stuart Shieber. Finale’s machine learning class was the first time I felt that I really “got” variational inference—a foundation for much of the work presented in this thesis—and Stuart’s endless knowledge of grammatical formalisms (and pretty much everything else) was invaluable in intellectually centering my later work on grammar induction.

I would like to give special thanks to: David Sontag, for his mentorship and guidance during my time at NYU and at Harvard; Slav Petrov, for his wonderful class at NYU which introduced me to the world of NLP; and Sam Wiseman, for many stimulating and insightful discussions.

More broadly, I am thankful to the many collaborators and mentors with whom I had the chance to work with and learn from: Allen Schmaltz, Yuntian Deng, Justin Chiu, Andrew Miller, Kelly Zhang, Demi Guo, and Carl Denton at Harvard; Jake Zhao, Yann LeCun, Yi-I Chiu, Kentaro Hanaki, and Darshan Hegde at NYU; Adji Bousso Dieng and David Blei at Columbia; Daniel Andor and Livio Baldini Soares at Google; Marc’Aurelio Ranzato, Laurens van der Maaten, and Maximilian Nickel at Facebook; and Chris Dyer, Gábor Melis, Lei Yu, and Adhiguna Kuncoro at DeepMind. I would also like to thank the other past and present members of the NLP group at Harvard: Sebastian Gehrmann, Zack Ziegler, Jeffrey Ling, Alex Wang, and Rachit Singh. I am especially grateful to Yonatan Belinkov, David Parkes, Mirac Suzgun, and Kyunghyun Cho for their guidance and advice during my final year when I was looking for a job.

Finally, I thank my family for their support throughout this journey.

1

Introduction

Understanding natural language involves complex underlying processes by which meaning is extracted from surface form. One approach to operationalizing such phenomena in computational models of natural language is through *probabilistic latent variable models*, which provide a modular, probabilistic framework for specifying prior knowledge and structural relationships in complex datasets. Latent variable models have a long and rich history in natural language processing, having contributed

to fundamental advances such as statistical alignment for translation (Brown et al., 1993), topic modeling (Blei et al., 2003), unsupervised part-of-speech tagging (Brown et al., 1992), and unsupervised parsing (Klein & Manning, 2004), among others. *Deep learning*, on the other hand, offers an alternative computational approach to modeling natural language through end-to-end learning of expressive, global models, where any phenomena necessary for the task are captured implicitly within the hidden layers of a neural network. Some major successes of deep learning in natural language processing include language modeling (Bengio et al., 2003; Mikolov et al., 2010; Zaremba et al., 2014; Radford et al., 2019), machine translation (Cho et al., 2014b; Sutskever et al., 2014; Bahdanau et al., 2015; Vaswani et al., 2017), question answering (Seo et al., 2017; Xiong et al., 2017; Wang & Jiang, 2017; Chen et al., 2017a), and more recently, representation learning for natural language understanding tasks through deep networks pretrained on self-supervised objectives (Peters et al., 2018; Devlin et al., 2018).

There has been much recent, exciting work on combining the complementary strengths of latent variable models and deep learning. Latent variable modeling makes it easy to explicitly specify model constraints through conditional independence properties, while deep learning makes it possible to parameterize these conditional likelihoods with powerful function approximators. This thesis investigate these “deep latent variable models” for natural language applications. We explore a variety of deep latent variable models that target different language phenomena, from continuous latent variable models of sentences (chapter 3) to attention in machine translation (chapter 4) to parse trees (chapter 5) and grammars (chapter 6). Core applications considered in this thesis include language modeling, machine translation, and unsupervised parsing.

While these deep latent variable models provide a rich, flexible framework for modeling many natural language phenomena, difficulties exist: deep parameterizations of conditional likelihoods usually make inference intractable, and latent variable objectives often complicate backpropagation by introducing points of stochasticity into a model’s computation graph. This thesis explores these issues at depth through the lens of *variational inference* (Jordan et al., 1999; Wainwright & Jordan, 2008), a key technique for performing approximate inference. We focus on a particular family of techniques called *amortized variational inference*, which trains a global inference network to output the parameters of an approximate posterior distribution given a set of variables to be conditioned upon (Kingma & Welling, 2014; Rezende et al., 2014; Mnih & Gregor, 2014). We show that a straightforward application of amortized variational inference is often insufficient for many applications of interest, and consider several extensions to the standard approach that lead to improved learning and inference.

1.1 THESIS OUTLINE

Each chapter begins by introducing a latent variable approach to modeling a particular aspect of language, which will bring about its own unique challenges in inference and learning. We then develop and study techniques for addressing each of these challenges that we anticipate will be more broadly applicable to other domains of interest. Concretely, the chapters are organized as follows:

- Chapter 2 gives a brief overview of latent variable models, exact and approximate inference, and the neural network machinery used throughout the thesis.
- Chapter 3 explores a continuous latent variable model of sentences with a fully

autoregressive generative model. We study a common failure mode in such models known as posterior collapse, and propose an improved, semi-amortized approach to approximate inference that is able to mitigate it.

- Chapter 4 provides a latent variable formulation of attention in neural machine translation, motivated by alignment in traditional statistical machine translation systems. We experiment with continuous relaxation techniques in addition to more traditional approaches for learning such models.
- Chapter 5 considers the problem of learning syntax-based language models where the latent space corresponds to the set of parse trees for a sentence. We show that posterior regularization through a structured inference network provides the appropriate inductive bias to facilitate the emergence of meaningful tree structures.
- Chapter 6 revisits grammar induction with contemporary parameterization and inference techniques. We combine classical dynamic programming algorithms with amortized variational inference and show that this collapsed variational inference approach can train richer grammars that go beyond the traditional context-free assumptions.
- Finally, chapter 7 concludes and discusses future outlook.

1.2 RELATED PUBLICATIONS

Portions of this thesis appeared in the following publications:

- **Chapters 1, 2, 7:** Y. Kim, S. Wiseman, A.M. Rush. “A Tutorial on Deep Latent Variable Models of Natural Language,” EMNLP Tutorial, 2018.

- **Chapter 3:** Y. Kim, S. Wiseman, A.C. Miller, D. Sontag, A.M. Rush. “Semi-Amortized Variational Autoencoders,” In Proceedings of ICML, 2018.
- **Chapter 4:** Y. Deng, Y. Kim, J. Chiu, D. Guo, A.M. Rush. “Latent Alignment and Variational Attention,” In Proceedings of NeurIPS, 2018.
- **Chapter 5:** Y. Kim, A.M. Rush, A. Kuncoro, C. Dyer, G. Melis. “Unsupervised Recurrent Neural Network Grammars,” In Proceedings of NAACL, 2019.
- **Chapter 6:** Y. Kim, C. Dyer, A.M. Rush. “Compound Probabilistic Context-Free Grammars for Grammar Induction,” In Proceedings of ACL, 2019.

2

Background

2.1 MOTIVATION

A *probabilistic latent variable model* specifies a joint distribution $p(x, z)$ over unobserved, latent variables z and observed variables x . Through a factorization of the joint distribution into modular components, it becomes possible to express rich structural relationships and reason about observed and unobserved factors of variation in

Portions of this chapter appeared in [Kim et al. \(2018b\)](#).

complex datasets within a probabilistic framework. Probabilistic latent variable models have a long and rich history in natural language processing. Influential applications of latent variable models include: part-of-speech induction with hidden Markov models (Merialdo, 1994), word alignment from parallel corpora in statistical machine translation (Brown et al., 1993), unsupervised morpheme discovery with latent segmentation models (Creutz & Lagus, 2002), topic modeling with latent Dirichlet allocation (Blei et al., 2003), unsupervised parsing with the constituent-context model (Klein & Manning, 2002) and dependency model with valence (Klein & Manning, 2004), and supervised parsing with latent variable probabilistic context-free grammars (Petrov et al., 2006). A core goal of latent variable modeling is often *structure discovery*. For example, successful induction of formal linguistic structures such as generative grammars can yield scientific insights as to the underlying processes that govern language acquisition, while thematic structures discovered through topic modeling can organize and summarize large amounts of text data.

Deep learning, broadly construed, describes a set of tools and models for end-to-end learning via numerical optimization against a predictive task. Key successes of deep learning in natural language processing include: language modeling (Bengio et al., 2003; Mikolov et al., 2010; Zaremba et al., 2014; Radford et al., 2019), machine translation (Cho et al., 2014b; Sutskever et al., 2014; Bahdanau et al., 2015; Vaswani et al., 2017), summarization (Rush et al., 2015; Cheng & Lapata, 2016; Nallapati et al., 2016; See et al., 2017), question answering (Seo et al., 2017; Xiong et al., 2017; Wang & Jiang, 2017; Chen et al., 2017a), text classification (Socher et al., 2013b; Kalchbrenner et al., 2014; Kim, 2014; Tai et al., 2015), representation learning through self-supervised objectives (Mikolov et al., 2013; Le & Mikolov, 2014; Peters et al., 2018; Devlin et al., 2018), and classical NLP tasks such as tagging (Collobert et al.,

2011; Lample et al., 2016; Ma & Hovy, 2016; He et al., 2017; Strubell et al., 2018) and parsing (Chen & Manning, 2014; Vinyals et al., 2015; Dyer et al., 2015; Kitaev & Klein, 2018). A primary goal of deep learning is generally *predictive accuracy*; we are interested in learning expressive, global models that are rich enough to model the underlying data and at the same time have the right inductive biases such that they generalize well to unseen examples.

In this thesis we study *deep latent variable models*, which combine the modularity of probabilistic latent variable models with the flexible modeling capabilities of deep networks. The overarching goal of deep latent variable models is to accurately model the underlying data with deep learning while at the same time exploiting latent variables for model interpretability, transparency, controllability, and structure discovery. The term “deep” in deep latent variable models refers to both (1) deep parameterizations of conditional distributions within latent variable models and (2) the use of deep neural networks to perform *approximate inference* over latent variables. As we will shortly see, these two uses of the term are connected: flexible parameterizations of distributions over high dimensional data enabled by deep networks generally lead to models in which exact posterior inference is intractable, which subsequently require the use of a separate *inference network* to perform approximate posterior inference in the model.

We begin this background chapter by briefly reviewing neural networks and latent variable models. We then discuss learning and inference in these models, both in the case where exact inference over the latent variables is tractable and when it is not. We conclude with an exposition of amortized variational inference and variational autoencoders, a central framework for learning deep latent variable models.

2.2 NOTATION

We will generally use x to denote the observed data and z to refer to the unobserved latent variables. In this thesis x is usually a sequence of discrete tokens,

$$x = [x_1, \dots, x_T]$$

where each $x_i \in \mathcal{V}$. Here $\mathcal{V} = \{1, \dots, V\}$ is a finite vocabulary set of size V , and T is the sequence length. The latent variable z can be a continuous vector (chapter 3), a categorical variable (chapter 4), a binary tree (chapter 5), or a combination thereof (chapter 6). Both x, z are random variables (i.e., measurable functions from sample space Ω to \mathbb{R}^n), though x will almost always be observed.

We use $p_x(y; \theta)$ to refer to the mass/density function of the random variable x parameterized by θ evaluated at y . When there is no ambiguity with regard to the random variable over which the distribution is induced, we will often drop the subscript on the mass/density function, or use a different letter (e.g. q instead of p for variational distributions). We overload p in two ways. First, we use p to more generally refer to a distribution over a random variable, e.g. when $x^{(n)}$ is sampled from a distribution $p(x; \theta)$,

$$x^{(n)} \sim p(x; \theta).$$

This distinction is in practice a mere formality since we always characterize distributions with their density/mass functions. The second use of p is to refer to the proba-

bility of an event, e.g. if x is discrete,

$$p(x = y; \theta) = p_x(y; \theta).$$

In order to simplify notation, we also overload x to additionally refer to the realization of the random variable. For example, when we use $\mathbb{E}_{p(x)}[f(x)]$ to refer to the expectation of a function $f(x)$,

$$\mathbb{E}_{p(x)}[f(x)] = \sum_x p(x) f(x),$$

we use x in the random variable sense in “ $\mathbb{E}_{p(x)}[f(x)]$ ” and in the realization sense in “ $\sum_x p(x) f(x)$ ”. The entropy of a distribution $p(x)$ is

$$\mathbb{H}[p(x)] = \mathbb{E}_{p(x)} [-\log p(x)],$$

and the Kullback–Leibler divergence between two distributions $p(x)$ and $q(x)$ is

$$\text{KL}[p(x) \parallel q(x)] = \mathbb{E}_{p(x)} \left[\log \frac{p(x)}{q(x)} \right].$$

We often use bold letters (e.g. \mathbf{x} instead of x and \mathbf{z} instead of z) to emphasize the fact that we are working with vectors/sequences. Vector concatenation between two vectors \mathbf{u} , \mathbf{v} is denoted as $[\mathbf{u}; \mathbf{v}]$. We will index vectors/matrices with square brackets or subscripts, e.g. $\mathbf{h}[i]$ or \mathbf{h}_i for the i -th element of a vector \mathbf{h} , and $\mathbf{W}[i]$ or \mathbf{W}_i is the i -th row of a matrix \mathbf{W} . Bracketed superscripts are used to index different data points, e.g. $x^{(1:N)} = \{x^{(n)}\}_{n=1}^N = \{x^{(1)}, \dots, x^{(N)}\}$ for a corpus of N sentences.

2.3 NEURAL NETWORKS

We now briefly introduce the neural network machinery to be used throughout the thesis. Neural networks are parameterized nonlinear functions, which transform an input vector \mathbf{e} into features \mathbf{h} using parameters θ . For example, a multilayer perceptron (MLP) computes features as follows:

$$\mathbf{h} = \text{MLP}(\mathbf{e}; \theta) = \mathbf{V}f(\mathbf{W}\mathbf{e} + \mathbf{b}) + \mathbf{a},$$

where f is an element-wise nonlinearity, such as tanh, ReLU, or sigmoid functions, and the set of neural network parameters is given by $\theta = \{\mathbf{V}, \mathbf{W}, \mathbf{a}, \mathbf{b}\}$. In practice multilayer perceptrons are often augmented with residual layers (He et al., 2015), batch normalization (Ioffe & Szegedy, 2015), layer normalization (Ba et al., 2016), and other modifications. We also make use of sequential neural networks (SeqNN) which operate over a sequence of input vectors $\mathbf{e}_{1:T} = [\mathbf{e}_1, \dots, \mathbf{e}_T]$ to output a sequence of features $\mathbf{h}_{1:T} = [\mathbf{h}_1, \dots, \mathbf{h}_T]$ as follows:

$$\mathbf{h}_{1:T} = \text{SeqNN}(\mathbf{e}_{1:T}; \theta),$$

For example, the classical Elman RNN (Elman, 1990) computes each \mathbf{h}_t as

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{e}_t + \mathbf{V}\mathbf{h}_{t-1} + \mathbf{b}),$$

where σ is the sigmoid function and $\theta = \{\mathbf{U}, \mathbf{V}, \mathbf{b}\}$. Commonly-employed sequential neural network architectures include long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997), gated recurrent units (GRU) (Cho et al., 2014b), and Trans-

formers (Vaswani et al., 2017). In this thesis we generally remain agnostic with regard to the particular architecture and simply treat $\text{MLP}(\cdot; \theta)$ and $\text{SeqNN}(\cdot; \theta)$ as (sub)differentiable functions with respect to θ , though for completeness we will often specify the architecture and hyperparameters when describing the experimental setup in each chapter.

As neural networks operate over vector representations of data, the first step in neural network-based approaches for natural language processing is to represent each word x_t with its vector representation \mathbf{e}_t . This is usually done through an embedding matrix,

$$\mathbf{e}_t = \mathbf{E}[x_t],$$

where \mathbf{E} is a $|\mathcal{V}| \times d$ matrix and $\mathbf{E}[x_t]$ is the word embedding corresponding to token x_t (i.e. x_t -th row of \mathbf{E}). It is also common practice to work with subword pieces as the atomic unit, for example characters or subword units obtained from byte pair encoding (Sennrich et al., 2016). Again we remain agnostic with regard to the level of atomicity and simply treat the input as a sequence of discrete tokens from a vocabulary \mathcal{V} .

Finally, neural networks make use of the *softmax* layer which applies an element-wise exponentiation to a vector $\mathbf{s} \in \mathbb{R}^n$ and renormalizes, i.e. $\mathbf{y} = \text{softmax}(\mathbf{s})$ means that each element of \mathbf{y} is given by

$$\text{softmax}(\mathbf{s})[k] = \mathbf{y}[k] = \frac{\exp(\mathbf{s}[k])}{\sum_{v=1}^K \exp(\mathbf{s}[v])}.$$

The softmax layer is often used to parameterize a distribution over a discrete space

over K elements, for example as the final output layer of a deep network or as an intermediate attention layer (chapter 4).

2.4 LATENT VARIABLE MODELS

A probabilistic latent variable model specifies a joint distribution $p(x, z; \theta)$ over the observed variable x and the unobserved variable z , where the distribution is parameterized by θ . For example, the Naive Bayes model assumes that a corpus of sentences, $\{x^{(n)}\}_{n=1}^N$, where each sentence $x^{(n)} = [x_1^{(n)}, \dots, x_T^{(n)}]$ has the same number of T tokens for simplicity, is generated according to the following probabilistic process:

1. For each sentence, sample latent variable $z^{(n)} \in \{1, \dots, K\}$ from a Categorical prior $p(z; \boldsymbol{\mu})$ with parameter $\boldsymbol{\mu} \in \Delta^{K-1}$. That is,

$$\begin{aligned} z^{(n)} &\sim p(z; \boldsymbol{\mu}) \\ p(z^{(n)} = k; \boldsymbol{\mu}) &= \boldsymbol{\mu}[k], \end{aligned}$$

where

$$\Delta^{K-1} = \left\{ \mathbf{s} : \sum_{k=1}^K \mathbf{s}[k] = 1, \mathbf{s}[k] \geq 0 \ \forall k \right\}$$

is the $K - 1$ -simplex.

2. Given $z^{(n)}$, sample each token $x_t^{(n)} \in \{1, \dots, V\}$ independently from a Categorical distribution with parameter $\boldsymbol{\pi}_z \in \Delta^{V-1}$. That is,

$$\begin{aligned} x_t^{(n)} &\sim p(x_t | z^{(n)}; \boldsymbol{\pi}) \\ p(x_t^{(n)} = v | z^{(n)} = k; \boldsymbol{\pi}) &= \boldsymbol{\pi}_k[v], \end{aligned}$$

where $\pi_k[v]$ is the probability of drawing word index v given that the latent variable $z^{(n)}$ takes on the value k .

Then, the probability of a sentence $x^{(n)} = [x_1^{(n)}, \dots, x_T^{(n)}]$ given $z^{(n)}$ is

$$p(x = x^{(n)} \mid z = z^{(n)}; \pi) = \prod_{t=1}^T \pi_{z^{(n)}}[x_t^{(n)}].$$

This generative process defines a factorization of the joint probability distribution of the entire corpus as

$$\begin{aligned} p(\{x^{(n)}, z^{(n)}\}_{n=1}^N; \pi, \mu) &= \prod_{n=1}^N p(z = z^{(n)}; \mu) p(x = x^{(n)} \mid z = z^{(n)}; \pi) \\ &= \prod_{n=1}^N \mu[z^{(n)}] \prod_{t=1}^T \pi_{z^{(n)}}[x_t^{(n)}]. \end{aligned}$$

The directed graphical model that delineates this generative process is shown in Figure 2.1, where we show random variables in circles (observed variables are shaded) and model parameters without circles.¹ It is clear that the graphical model specifies both the stochastic procedure for generating the data as well as the factorization of the joint distribution into conditional distributions. This model assumes that each token in x is generated independently, conditioned on z . This assumption is clearly naive (hence the name) but greatly reduces the number of parameters that need to be estimated. Letting $\theta = \{\mu, \pi\}$, the total number of parameters in this generative model is $K + KV$, where we have K parameters for μ and V parameters in π_z for

¹Graphical models provide a convenient framework for delineating the set of conditional independence assumptions that hold in a latent variable model. For example, directed graphical models factorize the joint distribution based on a top-down traversal of a graph. However, some latent variable models considered in this thesis (such as probabilistic context-free grammars in chapter 6) cannot conveniently be represented within the framework of graphical models.

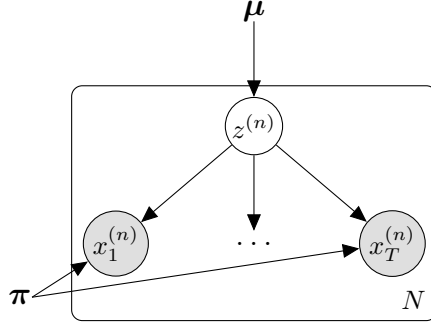


Figure 2.1: Graphical model for the Naive Bayes model. For simplicity, all sequences are depicted as having T tokens. All distributions are categorical, and the parameters are $\mu \in \Delta^{K-1}$ and $\pi = \{\pi_k \in \Delta^{V-1}\}_{k=1}^K$.

each of the K values of z .²

The Naive Bayes model is one of the simplest examples of a latent variable model of discrete sequences such as text. Intuitively, since each sentence $x^{(n)}$ comes with a corresponding latent variable $z^{(n)}$ governing its generation, we can see the $z^{(n)}$ values as inducing a clustering over the sentences $\{x^{(n)}\}_{n=1}^N$; sentences generated by the same value of $z^{(n)}$ belong to the same cluster. Throughout this thesis we will explore how different conditional independence assumptions in a latent variable model lead to different structures being encoded by the latent variables.

2.4.1 DEEP LATENT VARIABLE MODELS

Deep latent variable models parameterize the conditional distributions in a latent variable model with neural networks. For example, a “deep” Naive Bayes model might

²This model is overparameterized since we only need $V - 1$ parameters for a Categorical distribution over a set of size V . This is rarely an issue in practice.

parameterize $p(x \mid z; \theta)$ as follows:

$$p(x = x^{(n)} \mid z = z^{(n)}; \theta) = \prod_{t=1}^T \text{softmax}(\text{MLP}(\mathbf{z}^{(n)}; \theta))[x_t^{(n)}],$$

where $\mathbf{z}^{(n)}$ is the one-hot vector representation of $z^{(n)}$. Note that this modification does *not* change the underlying conditional independence assumptions; only the parameterization is changed. Concretely, whereas in the previous case we had a scalar $\pi_z[x_t]$ for each $z \in \{1, \dots, K\}$ and $x_t \in \{1, \dots, V\}$ (constrained that they are valid probabilities), we now parameterize π_z such that it is the output of a neural network. We will see in chapter 6 that probabilistic models with the same conditional independence assumptions but different parameterizations (i.e., scalar vs. neural parameterization) lead to different learning dynamics due to parameter sharing and other inductive biases from neural networks and the associated algorithms for optimizing them.

Another reason we are interested in deep latent variable models is that neural networks make it possible to define flexible distributions over high-dimensional data (such as text) without using too many parameters. As an example, let us now consider a sentence model similar to the Naive Bayes model, but which avoids the Naive Bayes assumption above (whereby each token is generated independently given z) using a sequential neural network such as an RNN. This will allow the probability of $x_t^{(n)}$ to depend on the entire history $x_{<t}^{(n)} = [x_1^{(n)}, \dots, x_{t-1}^{(n)}]$ of tokens preceding $x_t^{(n)}$. In

this deep variant, we might then define the probability of x given latent variable z as

$$\begin{aligned} p(x = x^{(n)} | z = z^{(n)}; \theta) &= \prod_{t=1}^T p(x_t = x_t^{(n)} | x_{<t} = x_{<t}^{(n)}, z = z^{(n)}; \theta) \\ &= \prod_{t=1}^T \text{softmax}(\mathbf{W}\mathbf{h}_{t-1}^{(n)})[x_t^{(n)}], \end{aligned}$$

where the hidden states \mathbf{h}_t are obtained as

$$\begin{aligned} \mathbf{h}_t^{(n)} &= \sigma(\mathbf{U}\mathbf{e}_t^{(n)} + \mathbf{V}\mathbf{h}_{t-1}^{(n)} + \mathbf{T}\mathbf{z}^{(n)} + \mathbf{b}), \\ \mathbf{e}_t^{(n)} &= \mathbf{E}[x_t^{(n)}]. \end{aligned}$$

That is, the probability distribution over x_t given z is given by a linear transformation over the hidden states produced by an Elman RNN that conditions on z through an additional matrix \mathbf{T} during each hidden state update. Here the parameters of the conditional distribution are given by $\theta = \{\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{T}, \mathbf{E}, \mathbf{b}\}$, and we show the corresponding graphical model in Figure 2.2. Note that unlike the deep Naive Bayes extension above, this is a more flexible probabilistic model with less stringent conditional independence assumptions. Further, this deep model allows for avoiding the Naive Bayes assumption while still using only $O(d^2 + Vd + Kd)$ parameters, assuming $\mathbf{h}_t^{(n)} \in \mathbb{R}^d$. On the other hand, a direct scalar parameterization of this model would require $O(KV^T)$ parameters, clearly highlighting the advantages afforded by neural network parameterizations.

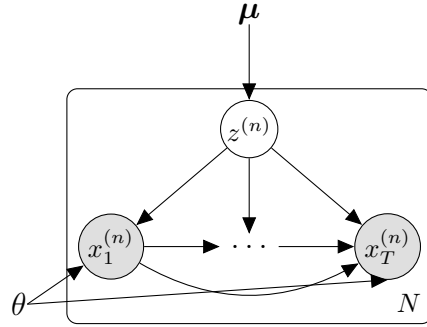


Figure 2.2: Graphical model representation of a categorical latent variable model with tokens generated by an RNN. For simplicity, all sequences are depicted as having T tokens. The $z^{(n)}$'s are drawn from a Categorical distribution with parameter μ , while $x^{(n)}$ is drawn from an RNN. The parameters of the RNN are given by $\theta = \{\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{T}, \mathbf{E}, \mathbf{b}\}$. See the text for more details.

2.5 LEARNING AND INFERENCE

After defining a latent variable model, we are interested in two related tasks: (1) we would like to be able to *learn* the parameters θ of the model based on observed data $x^{(1:N)}$, and (2) we would like to be able to perform *posterior inference* over the model. That is, we'd like to be able to compute the posterior distribution $p(z | x; \theta)$ (or approximations thereof) over the latent variables, given some data x . Note that posterior inference requires calculating the marginal distribution $p(x; \theta)$, since

$$p(z | x; \theta) = \frac{p(x, z; \theta)}{p(x; \theta)},$$

and hence we sometimes use “inference” to also refer to the calculation of the marginal distribution. As we will see, learning and inference are intimately connected because learning often uses inference as a subroutine.

2.5.1 MAXIMUM LIKELIHOOD ESTIMATION

The dominant approach to learning latent variable models in a probabilistic setting is to maximize the log marginal likelihood of observed data. This is equivalent to minimizing the KL-divergence between the true data distribution $p_\star(x)$ and the model distribution $p(x; \theta)$,

$$\text{KL}[p_\star(x) \parallel p(x; \theta)] = \mathbb{E}_{p_\star(x)} \left[\log \frac{p_\star(x)}{p(x; \theta)} \right]$$

where the latent variable z has been marginalized out, i.e.

$$p(x; \theta) = \sum_z p(x, z; \theta),$$

in the case that z is discrete. (The sum should be replaced with an integral if z is continuous). Assuming that each training example $x^{(n)}$ is sampled independently and identically from the true data distribution,

$$x^{(n)} \stackrel{\text{i.i.d.}}{\sim} p_\star(x),$$

maximum likelihood learning corresponds to the following optimization problem:

$$\begin{aligned} \arg \min_{\theta} \text{KL}[p_\star(x) \parallel p(x; \theta)] &= \arg \min_{\theta} \mathbb{E}_{p_\star(x)} \left[\log \frac{p_\star(x)}{p(x; \theta)} \right] \\ &= \arg \max_{\theta} \mathbb{E}_{p_\star(x)} [\log p(x; \theta)] \\ &\approx \arg \max_{\theta} \frac{1}{N} \sum_{n=1}^N \log p(x^{(n)}; \theta), \end{aligned}$$

where in the last line we approximate the expectation over $p_\star(x)$ with Monte Carlo samples from the data generating distribution (i.e. the training set).

2.5.2 TRACTABLE EXACT INFERENCE

We begin with the case where the log marginal likelihood, i.e.

$$\log p(x; \theta) = \log \sum_z p(x, z; \theta)$$

is tractable to evaluate. As mentioned above, this is equivalent to assuming posterior inference is tractable. In cases where $p(x, z; \theta)$ is parameterized by a deep model, the maximum likelihood problem,

$$\arg \max_{\theta} \sum_{n=1}^N \log p(x^{(n)}; \theta),$$

is usually not possible to solve for exactly. We will assume, however, that $p(x, z; \theta)$ is differentiable with respect to θ . The main tool for optimizing such models, then, is gradient-based optimization. In particular, define the log marginal likelihood over the training set $x^{(1:N)} = \{x^{(1)}, \dots, x^{(n)}\}$ as

$$\begin{aligned} L(\theta) &= \log p(x^{(1:N)}; \theta) \\ &= \sum_{n=1}^N \log p(x^{(n)}; \theta) \\ &= \sum_{n=1}^N \log \sum_z p(x^{(n)}, z; \theta). \end{aligned}$$

The gradient is given by

$$\begin{aligned}
\nabla_{\theta} L(\theta) &= \sum_{n=1}^N \frac{\nabla_{\theta} \sum_z p(x^{(n)}, z; \theta)}{p(x^{(n)}; \theta)} && \text{(chain rule)} \\
&= \sum_{n=1}^N \sum_z \frac{p(x^{(n)}, z; \theta)}{p(x^{(n)}; \theta)} \nabla_{\theta} \log p(x^{(n)}, z; \theta) && \text{(since } \nabla p(x, z) = p(x, z) \nabla \log p(x, z) \text{)} \\
&= \sum_{n=1}^N \mathbb{E}_{p(z|x^{(n)}; \theta)} [\nabla_{\theta} \log p(x^{(n)}, z; \theta)].
\end{aligned}$$

The above gradient expression involves an expectation over the posterior $p(z|x^{(n)}; \theta)$, and therefore is an example of how posterior inference is used as a subroutine in learning. With this expression for the gradient in hand, we may then learn by updating the parameters as

$$\theta^{(i+1)} = \theta^{(i)} + \eta \nabla_{\theta} L(\theta^{(i)}),$$

where η is the learning rate and $\theta^{(0)}$ is initialized randomly. In practice the gradient is calculated over *mini-batches* (i.e. random subsamples of the training set), and adaptive learning rates such as Adagrad (Duchi et al., 2011), Adadelata (Zeiler, 2012), and Adam (Kingma & Ba, 2015) are often used.

More generally, gradient ascent on the log marginal likelihood is an instance of an *Expectation-Maximization (EM)* algorithm (Dempster et al., 1977). The EM algorithm is an iterative method for learning latent variable models that admit tractable posterior inference. It maximizes a lower bound on the log marginal likelihood at each iteration. Given randomly-initialized starting parameters $\theta^{(0)}$, the algorithm updates the parameters via the following alternating procedure:

1. **E-step:** Derive the posterior under current parameters $\theta^{(i)}$, i.e., $p(z|x = x^{(n)}; \theta^{(i)})$ for all data points $n = 1, \dots, N$.

2. **M-step:** Define the *expected complete data likelihood* as

$$Q(\theta, \theta^{(i)}) = \sum_{n=1}^N \mathbb{E}_{p(z|x^{(n)}; \theta^{(i)})} [\log p(x^{(n)}, z; \theta)],$$

and then maximize this with respect to θ , holding $\theta^{(i)}$ fixed,

$$\theta^{(i+1)} = \arg \max_{\theta} Q(\theta, \theta^{(i)}).$$

It can be shown that EM improves the log marginal likelihood at each iteration, i.e.

$$\sum_{n=1}^N \log p(x^{(n)}; \theta^{(i+1)}) \geq \sum_{n=1}^N \log p(x^{(n)}; \theta^{(i)}).$$

In cases where finding the exact $\arg \max$ in the M-step is not possible (as will be the case with deep latent variable models), we can perform a gradient-based optimization step on Q . This highlights the connection between gradient ascent on the log marginal likelihood and EM, since

$$\begin{aligned} \nabla_{\theta} Q(\theta, \theta^{(i)}) &= \sum_{n=1}^N \mathbb{E}_{p(z|x^{(n)}; \theta^{(i)})} [\nabla_{\theta} \log p(x^{(n)}, z; \theta)] \\ &= \nabla_{\theta} L(\theta). \end{aligned}$$

This variant of EM is sometimes referred to as *generalized expectation maximization* (Dempster et al., 1977; Neal & Hinton, 1998; Murphy, 2012). The EM algorithm can in general be seen as performing coordinate ascent on a lower bound on the log marginal likelihood (Bishop, 2006). This view will become useful in the next section where we consider cases where exact posterior inference is intractable, and will moti-

vate *variational inference*, a class of methods which uses approximate but tractable posteriors in place of the true posterior.

2.5.3 VARIATIONAL INFERENCE

Previously we have considered cases in which posterior inference (or equivalently, calculation of the marginal likelihood) is tractable. Now we consider cases in which posterior inference is intractable. Variational inference (Hinton & van Camp, 1993; Jordan et al., 1999) is a technique for approximating an intractable posterior distribution $p(z|x;\theta)$ with a tractable surrogate. In the context of learning the parameters of a latent variable model, variational inference can be used in optimizing a lower bound on the log marginal likelihood that involves only an *approximate* posterior over latent variables, rather than the exact posteriors we have been considering until now.

We begin by defining a set of distributions \mathcal{Q} , known as the *variational family*, whose elements are distributions $q(z;\lambda)$ parameterized by λ (we only consider parametric families in this thesis). That is, \mathcal{Q} contains distributions over the latent variables z . We will use \mathcal{Q} to denote the entire variational family, and $q(z;\lambda) \in \mathcal{Q}$ to refer to a particular variational distribution within the variational family, which is picked out by λ . Working with continuous latent variable z , we now derive a lower bound on the log marginal likelihood $\log p(x;\theta) = \log \int_z p(x,z;\theta) dz$ that makes use of

the variational distribution $q(z; \lambda)$:

$$\begin{aligned}
& \log p(x; \theta) \\
&= \int q(z; \lambda) \log p(x; \theta) dz && \text{(since } \log p(x; \theta) \text{ is non-random)} \\
&= \int q(z; \lambda) \log \frac{p(x, z; \theta)}{p(z | x; \theta)} dz && \text{(rewriting } p(x; \theta)) \\
&= \int q(z; \lambda) \log \left(\frac{p(x, z; \theta)}{q(z; \lambda)} \frac{q(z; \lambda)}{p(z | x; \theta)} \right) dz && \text{(multiplying by 1)} \\
&= \int q(z; \lambda) \log \frac{p(x, z; \theta)}{q(z; \lambda)} dz + \int q(z; \lambda) \log \frac{q(z; \lambda)}{p(z | x; \theta)} dz && \text{(distribute log)} \\
&= \int q(z; \lambda) \log \frac{p(x, z; \theta)}{q(z; \lambda)} dz + \text{KL}[q(z; \lambda) \| p(z | x; \theta)] && \text{(definition of KL divergence)} \\
&= \mathbb{E}_{q(z; \lambda)} \left[\log \frac{p(x, z; \theta)}{q(z; \lambda)} \right] + \text{KL}[q(z; \lambda) \| p(z | x; \theta)] && \text{(definition of expectation)} \\
&= \text{ELBO}(\theta, \lambda; x) + \text{KL}[q(z; \lambda) \| p(z | x; \theta)] && \text{(definition of ELBO)} \\
&\geq \text{ELBO}(\theta, \lambda; x) && \text{(KL always non-negative)}
\end{aligned}$$

The above derivation shows that $\log p(x; \theta)$ is equal to a quantity called the *evidence lower bound*, or ELBO, plus the KL divergence between $q(z; \lambda)$ and the posterior distribution $p(z | x; \theta)$. Since the KL divergence is always non-negative, the ELBO is a lower-bound on $\log p(x; \theta)$, and it is this quantity that we attempt to maximize with variational inference.³

The form of the ELBO is worth looking at more closely. First, note that it is a function of θ, λ (the data x is fixed), and lower bounds the log marginal likelihood $\log p(x; \theta)$ for any λ . The bound is tight if the variational distribution equals the true

³Note that this derivation requires that the support of the variational distribution lie within the support of the true posterior, i.e., $p(z | x; \theta) = 0 \implies q(z; \lambda) = 0$ for all z . Otherwise, the second equality would have a division by zero. In contrast, we can have $q(z; \lambda) = 0$ and $p(z | x; \theta) > 0$ for some z , since the integral remains unchanged if we just integrate over the set $E = \{z : q(z; \lambda) > 0\}$.

posterior, i.e.

$$q(z; \lambda) = p(z | x; \theta) \implies \log p(x; \theta) = \text{ELBO}(\theta, \lambda; x).$$

It is also immediately evident that

$$\text{ELBO}(\theta, \lambda; x) = \log p(x; \theta) - \text{KL}[q(z; \lambda) \| p(z | x; \theta)].$$

In some scenarios the model parameters θ are given (and thus fixed), and the researcher is tasked with finding the best variational approximation to the true posterior. Under this setup, $\log p(x; \theta)$ is a constant with respect to λ and therefore maximizing the ELBO is equivalent to minimizing $\text{KL}[q(z; \lambda) \| p(z | x; \theta)]$. However for our purposes we are also interested in learning the generative model parameters θ .

Letting $x^{(1:N)} = \{x^{(1)}, \dots, x^{(N)}\}$ be the training set, the ELBO over the entire dataset is given by the sum of individual ELBOs,

$$\text{ELBO}(\theta, \lambda^{(1:N)}; x^{(1:N)}) = \sum_{n=1}^N \text{ELBO}(\theta, \lambda^{(n)}; x^{(n)}) = \sum_{n=1}^N \mathbb{E}_{q(z; \lambda^{(n)})} \left[\log \frac{p(x^{(n)}, z; \theta)}{q(z; \lambda^{(n)})} \right],$$

where the variational parameters are given by $\lambda^{(1:N)} = [\lambda^{(1)}, \dots, \lambda^{(n)}]$ (i.e. we have $\lambda^{(n)}$ for each data point $x^{(n)}$). Since $x^{(n)}$ are assumed to be drawn i.i.d, it is clear that the aggregate ELBO lower bounds the log likelihood of the training corpus,

$$\text{ELBO}(\theta, \lambda^{(1:N)}; x^{(1:N)}) \leq \log p(x^{(1:N)}; \theta).$$

It is this aggregate ELBO that we wish to maximize with respect to θ and $\lambda^{(1:N)}$ to train our model.

One possible strategy for maximizing the aggregate ELBO is coordinate ascent, where we maximize the objective with respect to the $\lambda^{(n)}$'s keeping θ fixed, then maximize with respect to θ keeping the $\lambda^{(n)}$'s fixed. In particular, we arrive at the following:

1. **Variational E-step:** For each $n = 1, \dots, N$, maximize the ELBO for each $x^{(n)}$ holding $\theta^{(i)}$ fixed,

$$\begin{aligned}\lambda^{(n)} &= \arg \max_{\lambda} \text{ELBO}(\theta^{(i)}, \lambda; x^{(n)}) \\ &= \arg \min_{\lambda} \text{KL}[q(z; \lambda^{(n)}) \parallel p(z \mid x^{(n)}; \theta^{(i)})],\end{aligned}$$

where the second equality holds since $\log p(x; \theta^{(i)})$ is a constant with respect to the $\lambda^{(n)}$'s.

2. **Variational M-step:** Maximize the aggregate ELBO with respect to θ holding the $\lambda^{(n)}$'s fixed,

$$\begin{aligned}\theta^{(i+1)} &= \arg \max_{\theta} \sum_{n=1}^N \text{ELBO}(\theta, \lambda^{(n)}; x^{(n)}) \\ &= \arg \max_{\theta} \sum_{n=1}^N \mathbb{E}_{q(z; \lambda^{(n)})} [\log p(x^{(n)}, z; \theta)],\end{aligned}$$

where the second equality holds since the $\mathbb{E}_{q(z; \lambda^{(n)})} [-\log q(z; \lambda^{(n)})]$ portion of the ELBO is constant with respect to θ .

This style of training is also known as *variational expectation maximization* (Neal & Hinton, 1998). In variational EM, the E-step, which usually performs exact posterior inference, is instead replaced with variational inference which finds the best varia-

tional approximation to the true posterior. The M-step maximizes the expected complete data likelihood where the expectation is taken with respect to the variational posterior distribution.

If we consider the case where the variational family is flexible enough to include the true posterior,⁴ then it is clear that the above reduces to the EM algorithm, since in the first step $\text{KL}[q(z; \lambda^{(n)}) \parallel p(z \mid x^{(n)}; \theta^{(i)})]$ is minimized when $q(z; \lambda^{(n)})$ equals the true posterior. Therefore, we can view EM as performing coordinate ascent on the ELBO where the variational family is arbitrarily flexible. Of course, this case is uninteresting since we have assumed that exact posterior inference is intractable. We are therefore interested in choosing a variational family that is flexible enough and at the same time allows for tractable optimization.

In practice, performing coordinate ascent on the entire dataset is usually too expensive. The variational E-step can instead be performed over mini-batches. As with generalized EM, the M-step can also be modified to perform gradient-based optimization. It is also possible to perform the E-step only approximately, again using gradient-based optimization. This style of approach leads to a class of methods called *stochastic variational inference (SVI)* (Hoffman et al., 2013). Concretely, for each $x^{(n)}$ in the mini-batch (of size B) we can randomly initialize $\lambda_0^{(n)}$ and perform gradient ascent on the ELBO with respect to λ for K steps,

$$\lambda_k^{(n)} = \lambda_{k-1}^{(n)} + \eta \nabla_{\lambda} \text{ELBO}(\theta, \lambda_k^{(n)}; x^{(n)}), \quad k = 1, \dots, K.$$

Then the M-step, which updates θ , proceeds with the variational parameters $\lambda_K^{(1)}, \dots, \lambda_K^{(B)}$

⁴That is, $\forall x \exists \lambda_x$ such that $q(z; \lambda_x) = p(z \mid x; \theta) \forall z$.

held fixed,

$$\theta^{(i+1)} = \theta^{(i)} + \eta \nabla_{\theta} \sum_{n=1}^B \mathbb{E}_{q(z|\lambda_K^{(n)})} \left[\log p(x^{(n)}, z; \theta^{(i)}) \right].$$

Variational inference is a rich field of active research, and we have only covered a small portion of it in this section. For example, we have not covered *coordinate ascent variational inference*, which allows for closed-form updates in the E-step for conditionally conjugate models. We refer the interested reader to [Wainwright & Jordan \(2008\)](#), [Blei et al. \(2017\)](#), and [Zhang et al. \(2017\)](#) for further reading.

2.5.4 AMORTIZED VARIATIONAL INFERENCE

The variational E-step requires that we (approximately) find the best variational parameters $\lambda^{(n)}$ for each $x^{(n)}$. Even in mini-batch settings, this optimization procedure can be expensive, especially if a closed-form update is not available, which is typically the case in deep latent variable models. In such cases, one could rely on iterative methods to find approximately optimal variational parameters as in SVI (see previous section), but this may still be prohibitively expensive since each gradient calculation $\nabla_{\lambda} \text{ELBO}(\theta, \lambda; x^{(n)})$ requires backpropagating gradients through the generative model.

As an alternative, we can instead *predict* the variational parameters by applying a neural network, called an *inference network*,⁵ to the input $x^{(n)}$ for which we would like to calculate an approximate posterior:

$$\lambda^{(n)} = \text{enc}(x^{(n)}; \phi).$$

The inference network is trained to perform variational inference for all the data

⁵Also referred to as a *recognition network* or an *encoder*.

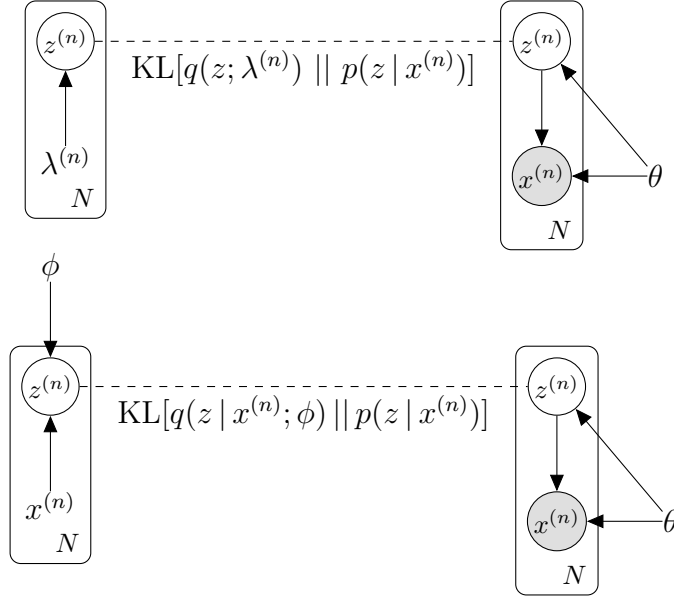


Figure 2.3: (Top) Traditional variational inference uses variational parameters $\lambda^{(n)}$ for each data point $x^{(n)}$. (Bottom) Amortized variational inference employs a global inference network ϕ that is run over the input $x^{(n)}$ to produce the local variational distributions.

points, i.e.

$$\max_{\phi} \sum_{n=1}^N \text{ELBO}(\theta, \text{enc}(x^{(n)}; \phi); x^{(n)}).$$

The inference network parameters ϕ is itself optimized with gradient ascent. Importantly, the same encoder network can be used for all $x^{(n)}$ we are interested in, and it is therefore unnecessary to optimize separate $\lambda^{(n)}$ for each $x^{(n)}$ we encounter. This style of inference is also known as *amortized variational inference (AVI)* (Kingma & Welling, 2014; Rezende et al., 2014; Mnih & Gregor, 2014), as the task of performing approximate posterior inference is *amortized* across each optimization step and data point through the shared encoder.⁶ This is illustrated in Figure 2.3. AVI is usually

⁶Amortized inference is a more general concept and has been applied to a variety of settings, including structured prediction (Srikumar et al., 2012; Tu & Gimpel, 2018), Monte Carlo sampling (Paige & Wood, 2016), and Bethe free energy minimization (Wiseman & Kim, 2019).

Inference (E-step)	$\min_{q(z)} \text{KL}[q(z) \parallel p(z \mid x; \theta)]$
Exact posterior inference	$q(z) = p(z \mid x; \theta)$
Maximum a posteriori (MAP)	$q(z) = \mathbb{1}\{z = \arg \max_z p(z \mid x; \theta)\}$
Coord. ascent variational inference (CAVI)	$q(z_j) \propto \exp \left(\mathbb{E}_{q(z_{-j})} [\log p(x, z; \theta)] \right)$
Stochastic variational inference (SVI)	$q(z; \lambda), \quad \lambda = \lambda + \eta \nabla_{\lambda} \text{ELBO}(\theta, \lambda; x)$
Amortized variational inference (AVI)	$q(z; \lambda), \quad \lambda = \text{enc}(x; \phi)$
Learning (M-step)	$\max_{\theta} \mathbb{E}_{q(z)} [\log p(x, z; \theta)]$
Analytic update	$\theta = \arg \max_{\theta} \mathbb{E}_{q(z)} [\log p(x, z; \theta)]$
Gradient-based update	$\theta = \theta + \eta \mathbb{E}_{p(z \mid x; \theta)} [\nabla_{\theta} \log p(x, z; \theta)]$
Other approximations	$\theta \approx \arg \max_{\theta} \mathbb{E}_{q(z)} [\log p(x, z; \theta)]$

Table 2.1: A simplified landscape of the different optimization methods for training latent variable models with maximum likelihood learning. The “Expectation” or “Inference” step (E-step) corresponds to performing posterior inference, i.e. minimizing $\text{KL}[q(z) \parallel p(z \mid x; \theta)]$. The “Maximization” or “Learning” step (M-step) corresponds to maximizing the complete data likelihood under the inferred posterior, i.e. $\mathbb{E}_{q(z)} [\log p(x, z; \theta)]$.

much faster than both SVI and traditional VI, as one can simply run the inference network over $x^{(n)}$ to obtain the variational parameters, which should approximate the true posterior well if the inference network is sufficiently expressive and well-trained.

SUMMARY Table 2.1 gives a greatly simplified overview of the different methods for training latent variable models with maximum likelihood learning. At a high level, many methods can be viewed as performing iterative optimization which alternates between performing an update on the auxiliary variational parameters (E-step) and an update on the model parameters under the inferred posterior (M-step). Table 2.2 shows how different combinations of E- and M-steps lead to different, commonly-utilized methods for learning latent variable models.

Method	E-step	M-step
Expectation maximization	Exact posterior	Analytic
Log marginal likelihood	Exact posterior	Gradient
Hard EM	MAP	Analytic
Variational EM	CAVI/SVI	Analytic/Gradient/Other
Variational autoencoder	AVI	Gradient

Table 2.2: Different combinations of E-step and M-step lead to different methods for learning latent variable models.

2.5.5 VARIATIONAL AUTOENCODERS

Training deep latent variable models with amortized variational inference leads to a family of models called *variational autoencoders (VAE)* (Kingma & Welling, 2014).⁷ For latent variable models that factorize as

$$p(x, z; \theta) = p(x | z; \theta)p(z; \theta),$$

we can rearrange the ELBO as follows:

$$\begin{aligned}
\text{ELBO}(\theta, \phi; x) &= \mathbb{E}_{q(z | x; \phi)} \left[\log \frac{p(x, z; \theta)}{q(z; \phi)} \right] \\
&= \mathbb{E}_{q(z | x; \phi)} \left[\log \frac{p(x | z; \theta)p(z; \theta)}{q(z; \phi)} \right] \\
&= \mathbb{E}_{q(z | x; \phi)} [\log p(x | z; \theta)] - \text{KL} [q(z | x; \phi) \| p(z; \theta)].
\end{aligned}$$

Above, for brevity we have written $\text{ELBO}(\theta, \phi; x)$ in place of $\text{ELBO}(\theta, \text{enc}(x; \phi); x)$, and $q(z | x; \phi)$ in place of $q(z; \text{enc}(x; \phi))$, and we will use this notation going forward.

The *autoencoder* part of the name stems from the fact that and the first term in the

⁷While the term “variational autoencoder” is widely-used, it is somewhat less than ideal since it does not explicitly separate the *model* (i.e. the underlying generative model) from *inference* (i.e. how one performs inference in the model).

above derivation is the expected reconstruction likelihood of x given the latent variables z , which is roughly equivalent to an autoencoding objective. The second term can be viewed as a regularization term that pushes the variational distribution to be close to the prior.

GRADIENTS In the standard VAE setup, the inference network and the generative model are jointly trained by maximizing the ELBO with gradient ascent:

$$\begin{aligned}\theta^{(i+1)} &= \theta^{(i)} + \eta \nabla_{\theta} \text{ELBO}(\theta^{(i)}, \phi^{(i)}; x^{(n)}) \\ \phi^{(i+1)} &= \phi^{(i)} + \eta \nabla_{\phi} \text{ELBO}(\theta^{(i)}, \phi^{(i)}; x^{(n)}).\end{aligned}$$

The above update uses a fixed learning rate for a single data point, but in practice adaptive learning rates and mini-batches are used. Note that unlike the coordinate ascent-style training from previous section, θ and ϕ are trained together end-to-end.

We now derive the gradient expressions for both θ and ϕ . The gradient of the ELBO with respect to θ is given by

$$\nabla_{\theta} \text{ELBO}(\theta, \phi; x) = \mathbb{E}_{q(z|x;\phi)}[\nabla_{\theta} \log p(x, z; \theta)],$$

where we can push the gradient inside the expectations since the distribution over which we are taking the expectation does not depend on θ .⁸ For the gradient with

⁸This is valid under the assumption that we can differentiate under the integral sign (i.e. swap the gradient/integral signs), which holds under mild conditions, e.g. conditions which satisfy the hypotheses of the dominated convergence theorem.

respect to ϕ , we first separate the ELBO into two parts,

$$\begin{aligned}\nabla_{\phi} \text{ELBO}(\theta, \phi; x) &= \nabla_{\phi} \mathbb{E}_{q(z|x; \phi)} \left[\log \frac{p(x, z; \theta)}{q(z|x; \phi)} \right] \\ &= \nabla_{\phi} \mathbb{E}_{q(z|x; \phi)} [\log p(x, z; \theta)] - \nabla_{\phi} \mathbb{E}_{q(z|x; \phi)} [\log q(z|x; \phi)].\end{aligned}$$

Unlike the case with θ , we cannot simply push the gradient sign inside the expectation since the distribution with which we are taking the expectation depends on ϕ .

We derive the gradients of the two terms in the above expression separately. The first term involves the *score function gradient estimator* (Glynn, 1987; Williams, 1992; Fu, 2006),

$$\begin{aligned}\nabla_{\phi} \mathbb{E}_{q(z|x; \phi)} [\log p(x, z; \theta)] &= \nabla_{\phi} \int \log p(x, z; \theta) \times q(z|x; \phi) dz \\ &= \int \log p(x, z; \theta) \times \nabla_{\phi} q(z|x; \phi) dz && \text{(differentiate under integral)} \\ &= \int \log p(x, z; \theta) \times q(z|x; \phi) \times \nabla_{\phi} \log q(z|x; \phi) dz && \text{(since } \nabla q = q \nabla \log q) \\ &= \mathbb{E}_{q(z|x; \phi)} [\log p(x, z; \theta) \times \nabla_{\phi} \log q(z|x; \phi)].\end{aligned}$$

The second term is given by,

$$\begin{aligned}
& \nabla_\phi \mathbb{E}_{q(z|x;\phi)}[\log q(z|x;\phi)] \\
&= \nabla_\phi \int \log q(z|x;\phi) \times q(z|x;\phi) dz \\
&= \int \nabla_\phi \left(\log q(z|x;\phi) \times q(z|x;\phi) \right) dz && \text{(differentiate under integral)} \\
&= \int q(z|x;\phi) \nabla_\phi \log q(z|x;\phi) + \log q(z|x;\phi) \nabla_\phi q(z|x;\phi) dz && \text{(product rule)} \\
&= \int \nabla_\phi q(z|x;\phi) dz + \int \log q(z|x;\phi) \times \nabla_\phi q(z|x;\phi) dz && \text{(apply } \nabla q = q \nabla \log q) \\
&= 0 + \int \log q(z|x;\phi) \times \nabla_\phi q(z|x;\phi) dz && \left(\int \nabla q = \nabla \int q = \nabla 1 = 0 \right) \\
&= \int \log q(z|x;\phi) \times q(z|x;\phi) \times \nabla_\phi \log q(z|x;\phi) && \text{(apply } \nabla q = q \nabla \log q \text{ again)} \\
&= \mathbb{E}_{q(z|x;\phi)}[\log q(z|x;\phi) \times \nabla_\phi \log q(z|x;\phi)]
\end{aligned}$$

Putting it all together, we have

$$\nabla_\phi \text{ELBO}(\theta, \phi; x) = \mathbb{E}_{q(z|x;\phi)} \left[\log \frac{p(x, z; \theta)}{q(z|x;\phi)} \times \nabla_\phi \log q(z|x;\phi) \right],$$

which is reminiscent of policy gradient-style reinforcement learning with reward given by

$$\log \frac{p(x, z; \theta)}{q(z|x;\phi)}.$$

This expectation can again be estimated with Monte Carlo samples. While the Monte Carlo gradient estimator is unbiased it will suffer from high variance, and a common strategy to reduce the variance is to use a *control variate* B . First observe that we

can subtract B from the reward and not affect the gradient, i.e.

$$\nabla_{\phi} \text{ELBO}(\theta, \phi; x) = \mathbb{E}_{q(z|x;\phi)} \left[\left(\log \frac{p(x, z; \theta)}{q(z|x;\phi)} - B \right) \nabla_{\phi} \log q(z|x;\phi) \right],$$

since

$$\begin{aligned} \mathbb{E}_{q(z|x;\phi)}[B \times \nabla_{\phi} \log q(z|x;\phi)] &= \int B \times q(z|x;\phi) \times \nabla_{\phi} \log q(z|x;\phi) dz \\ &= \int B \times \nabla_{\phi} q(z|x;\phi) dz \\ &= \nabla_{\phi} \int B \times q(z|x;\phi) dz \\ &= 0 \end{aligned}$$

as long as B does not depend on the latent variable z or ϕ . We can then estimate the above gradient instead with Monte Carlo samples. Common techniques for B include a running average of previous rewards, a learned function (Mnih & Gregor, 2014), or combinations thereof. In chapters 4 and 5 we experiment with different control variates for training such models.

PATHWISE GRADIENT ESTIMATORS The above expression for the gradient with regard to ϕ is agnostic with regard to the family of variational distributions $q(z|x;\phi)$. Now let us derive an alternative gradient estimator which *does* assume a particular family. In particular, suppose our variational posterior family is multivariate Gaussian with diagonal covariance matrix, i.e.,

$$q(z|x;\phi) = \mathcal{N}(z; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)),$$

where the parameters are again given by the inference network

$$\boldsymbol{\mu}, \boldsymbol{\sigma}^2 = \text{enc}(x; \phi).$$

(In general we use $\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ to refer to the density of a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$). In this case we can exploit this choice of variational family to derive another estimator. In particular, we observe that our variational family of Gaussian distributions is *reparameterizable* (Kingma & Welling, 2014; Rezende et al., 2014; Glasserman, 2013) in the sense that we can obtain a sample from the variational posterior by sampling from a base noise distribution and applying a deterministic transformation,

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I}), \quad z = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon}.$$

Observe that z remains distributed according to $\mathcal{N}(z; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$, but we may now express the gradient with respect to ϕ as

$$\begin{aligned} \nabla_{\phi} \mathbb{E}_{q(z|x;\phi)} \left[\log \frac{p(x, z; \theta)}{q(z|x;\phi)} \right] &= \nabla_{\phi} \mathbb{E}_{\mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I})} \left[\log \frac{p(x, z = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon}; \theta)}{q(z = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon}|x;\phi)} \right] \\ &= \mathbb{E}_{\mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I})} \left[\nabla_{\phi} \log \frac{p(x, z = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon}; \theta)}{q(z = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon}|x;\phi)} \right], \end{aligned}$$

where the second equality follows since the expectation no longer depends on ϕ .

The estimator derived from this *reparameterization trick* just discussed is called a *pathwise gradient estimator*, and empirically yields much lower-variance estimators compared to the score function gradient estimator.⁹ Intuitively, the pathwise gradient

⁹However, there are cases where the score function gradient estimator has lower variance. See, for example, Mohamed et al. (2019).

Ch.	Latent Variable	Generative Model	Learning & Inference
3	Continuous vector	Autoregressive neural language model	Semi-Amortized VI (amortized + stochastic VI)
4	Categorical variable	Sequence-to-sequence model with attention	Continuous relaxation with Gumbel-Softmax
5	Unlabeled binary parse tree	Recurrent neural network grammars	Posterior regularization with CRF inference network
6	Grammar & Continuous vector	Compound probabilistic context-free grammars	Collapsed amortized VI with dynamic programming

Table 2.3: Summary of different latent variables, generative models, and learning & inference techniques explored in this thesis.

estimator “differentiates through” the generative model and therefore has more information than the score function gradient estimator, which treats the generative model as a black-box reward function. In this thesis, we utilize the pathwise estimator when possible, though we will also study latent variable models in which the reparameterization trick is not straightforwardly applicable, for example when z is discrete.

2.6 THESIS ROADMAP

This thesis studies different latent variable models that target various language phenomena. Table 2.3 summarizes the different types of latent variables, generative models, and learning/inference techniques explored in this thesis.

3

Latent Variable Model of Sentences & Semi-Amortized Variational Inference

3.1 INTRODUCTION

In this chapter we consider a continuous latent variable model of text where we assume that each sentence/paragraph is generated from a continuous vector. The gener-

The material in this chapter is adapted from [Kim et al. \(2018a\)](#).

ative model parameterizes the probability distribution over the next word through an autoregressive neural language model that composes the sentence-level latent variable with representations of previously-generated tokens (Bowman et al., 2016), similar to the example model we saw in 2.4.1. One motivation for this type of approach is to model global properties of sentences with a latent vector while simultaneously using an autoregressive model to target local properties. However, it is well known that a straightforward application of amortized variational inference to train such models results in a phenomenon known as *posterior collapse*, whereby the generative model does not make use of the latent variable, ultimately rendering it meaningless (Bowman et al., 2016; Yang et al., 2017). This chapter develops a semi-amortized inference approach that augments amortized variational inference with stochastic variational inference. We find that this technique is able to partially mitigate posterior collapse in variational autoencoders even when utilizing an expressive generative model.

3.2 BACKGROUND

We begin by reviewing the generative model from Bowman et al. (2016) and the reparameterization trick as it applies to this particular model. We then discuss an important issue called posterior collapse that arises when training latent variable models which utilize a fully-autoregressive generative model (3.2.2), as well as the amortization gap that results from suboptimal variational parameters obtained from a global inference network (3.2.3).

3.2.1 GENERATIVE MODEL

Let us revisit the variational autoencoder from 2.5.5 with a spherical Gaussian prior and a Gaussian variational family, see how it may be learned in practice with gradient-based optimization utilizing pathwise gradient estimators. In this chapter we work with the following generative model from Bowman et al. (2016),

- First sample latent vector $\mathbf{z} \sim \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$, $\mathbf{z} \in \mathbb{R}^d$.
- Sample each token sequentially as follows:

$$\begin{aligned} x_t &\sim p(x_t | x_{<t}, \mathbf{z}; \theta), \\ p(x_t | x_{<t}, \mathbf{z}; \theta) &= \text{softmax}(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}), \\ \mathbf{h}_t &= \text{LSTM}(\mathbf{h}_{t-1}, [\mathbf{e}_{x_t}; \mathbf{z}]). \end{aligned}$$

Concretely, the distribution over the next token is given by a softmax function over an affine transformation of a context vector \mathbf{h}_t , which is obtained through an LSTM network that conditions on the sentence-level latent variable \mathbf{z} by feeding it as additional input alongside the word embedding at each time step.

For learning θ , recall that we want to maximize the ELBO, which lower bounds the log marginal likelihood

$$\begin{aligned} \log p(x; \theta) &\geq \text{ELBO}(\theta, \phi; x) \\ &= \mathbb{E}_{q(\mathbf{z} | x; \phi)} \left[\log \frac{p(x, \mathbf{z}; \theta)}{q(\mathbf{z} | x; \phi)} \right] \\ &= \mathbb{E}_{q(\mathbf{z} | x; \phi)} [\log p(x | \mathbf{z}; \theta)] - \text{KL}[q(\mathbf{z} | x; \phi) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})]. \end{aligned}$$

Define the variational family \mathcal{Q} to be the set of Gaussian distributions with diagonal covariance matrices, whose parameters are predicted from an inference network. That is,

$$q(\mathbf{z} \mid x; \phi) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)),$$

where

$$\boldsymbol{\mu}, \boldsymbol{\sigma}^2 = \text{enc}(x; \phi), \quad \boldsymbol{\mu} \in \mathbb{R}^d, \boldsymbol{\sigma}^2 \in \mathbb{R}_{\geq 0}^d.$$

A popular parameterization for the inference network $\text{enc}(x; \phi)$ is

$$\mathbf{s}_{1:T} = \text{SeqNN}(\mathbf{e}_{1:T}; \phi),$$

$$\boldsymbol{\mu} = \text{MLP}_1(\mathbf{s}_T),$$

$$\log \boldsymbol{\sigma} = \text{MLP}_2(\mathbf{s}_T).$$

As we saw in 2.5.5, an estimator for the gradient with respect to θ is simple to obtain. For the gradient with respect to ϕ , we first observe that the generative process factorizes the joint distribution as

$$p(x, \mathbf{z}; \theta) = p(x \mid \mathbf{z}; \theta) \times \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}),$$

so we can also express the gradient of the ELBO with respect to ϕ as

$$\nabla_{\phi} \text{ELBO}(\theta, \phi; x) = \nabla_{\phi} \mathbb{E}_{q(\mathbf{z} \mid x; \phi)} [\log p(x \mid \mathbf{z}; \theta)] - \nabla_{\phi} \text{KL}[q(\mathbf{z} \mid x; \phi) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})].$$

Beginning with the second term, the KL divergence between a diagonal Gaussian and the spherical Gaussian has an analytic solution given by

$$\text{KL}[q(\mathbf{z} | x; \phi) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})] = -\frac{1}{2} \sum_{j=1}^d (\log \sigma_j^2 - \sigma_j^2 - \mu_j^2 + 1),$$

and therefore $\nabla_{\phi} \text{KL}[q(\mathbf{z} | x; \phi) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})]$ is easy to calculate. For the first term, we use the pathwise gradient estimator,

$$\begin{aligned} \nabla_{\phi} \mathbb{E}_{q(\mathbf{z} | x; \phi)} [\log p(x | \mathbf{z}; \theta)] &= \nabla_{\phi} \mathbb{E}_{\mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I})} [\log p(x | \boldsymbol{\mu} + \boldsymbol{\sigma} \epsilon; \theta)] \\ &= \mathbb{E}_{\mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I})} [\nabla_{\phi} \log p(x | \boldsymbol{\mu} + \boldsymbol{\sigma} \epsilon; \theta)], \end{aligned}$$

where we approximate the expectation with a single sample. We can then perform end-to-end gradient-based training with respect to both the generative model θ and the inference network ϕ .

3.2.2 POSTERIOR COLLAPSE

We now discuss an important issue which affects training of the text variational autoencoders with a fully autoregressive generative model. In the model introduced above, the likelihood model is allowed to fully condition on the entire history through the RNN’s state \mathbf{h}_t ,

$$\begin{aligned} \mathbf{h}_t &= \text{LSTM}(\mathbf{h}_{t-1}, [\mathbf{x}_{t-1}; \mathbf{z}]) \\ p(x_t = v | x_{<t}, \mathbf{z}; \theta) &= \text{softmax}(\mathbf{W} \mathbf{h}_t)[v], \end{aligned}$$

with the motivation for this approach being to capture global properties with \mathbf{z} . However, Bowman et al. (2016) observe that these types of models experience *posterior collapse* (or *latent variable collapse*), whereby the likelihood model $p(x | \mathbf{z}; \theta)$ ignores the latent variable and simply reduces to a non-latent variable language model. That is, x and \mathbf{z} become independent. Indeed, looking at the ELBO in more detail, we observe that if the distribution over x can be modeled *without* \mathbf{z} , the model is incentivized to make the variational posterior approximately equal to the prior,

$$\text{KL}[q(\mathbf{z} | x; \phi) \| p(\mathbf{z}; \gamma)] \approx 0,$$

regardless of how expressively one parameterizes $q(\mathbf{z} | x; \phi)$ (here γ parameterizes the prior). More formally, Chen et al. (2017b) show that this phenomenon may be justified under the “bits-back” argument: if the likelihood model is rich enough to model the true data distribution $p_\star(x)$ without using any information from \mathbf{z} , then the global optimum is obtained by setting

$$\begin{aligned} p(x | \mathbf{z}; \theta) &= p_\star(x) \\ p(\mathbf{z} | x; \theta) &= q(\mathbf{z} | x; \phi) = p(\mathbf{z}; \gamma). \end{aligned}$$

Since any distribution $p(x)$ can be factorized autoregressively as

$$p(x) = p(x_1) \prod_{t=2}^T p(x_t | x_{<t}),$$

it is possible that a richly parameterized deep network can model $p_\star(x)$ without relying on \mathbf{z} . As such, to avoid posterior collapse, past work has made conditional independence assumptions and instead used multilayer perceptrons (Miao et al., 2016,

2017) or convolutional networks (Yang et al., 2017; Semeniuta et al., 2017; Shen et al., 2018a) to parameterize the likelihood model, often at the cost of predictive accuracy (i.e. perplexity). In this chapter we consider an alternative approach which targets posterior collapse while still utilizing expressive generative models.

3.2.3 AMORTIZATION GAP

Another issue in variational autoencoders is the *amortization gap*, which arises from restricting the variational family to be the class of distributions whose parameters are obtainable by running a parameteric inference network over the input. While such a global parameterization allows for fast training/inference, it may be too strict of a restriction compared to methods such as stochastic variational inference (see 2.5.3) which obtain variational parameters for each datum via local optimization. In particular, letting λ^* be the best variational parameter for a given data point,

$$\lambda^* = \arg \min_{\lambda} \text{KL}[q(\mathbf{z}; \lambda) \parallel p(\mathbf{z} \mid x; \theta)]$$

we can break down the *inference gap*—the gap between the variational posterior from the inference network and the true posterior—as follows,

$$\underbrace{\text{KL}[q(\mathbf{z} \mid x; \phi) \parallel p(\mathbf{z} \mid x; \theta)]}_{\text{inference gap}} = \underbrace{\text{KL}[q(\mathbf{z}; \lambda^*) \parallel p(\mathbf{z} \mid x; \theta)]}_{\text{approximation gap}} + \underbrace{\text{KL}[q(\mathbf{z} \mid x; \phi) \parallel p(\mathbf{z} \mid x; \theta)] - \text{KL}[q(\mathbf{z}; \lambda^*) \parallel p(\mathbf{z} \mid x; \theta)]}_{\text{amortization gap}}.$$

Therefore the inference gap consists of two parts: the *approximation gap*, which is the gap between the true posterior and the best possible variational posterior within \mathcal{Q} ,

and the *amortization gap*, which quantifies the gap between inference network posterior and the best possible variational posterior. [Cremer et al. \(2018\)](#) observe that this amortization gap in practice can be large even for richly parameterized inference networks.

3.3 SEMI-AMORTIZED VARIATIONAL AUTOENCODERS

In this chapter we consider a method that combines amortized and stochastic variational inference to reduce the inference gap, which results in better training of generative models and partially addresses the posterior collapse phenomenon described above. In particular we propose a *semi-amortized* approach to training deep latent variable models which combines amortized variational inference (AVI) with stochastic variational inference (SVI). This semi-amortized variational autoencoder (SA-VAE) is trained using a combination of AVI and SVI steps:

1. Sample $x \sim p_\star(x)$
2. Set $\lambda_0 = \text{enc}(x; \phi)$
3. For $k = 0, \dots, K - 1$,

$$\lambda_{k+1} = \lambda_k + \alpha \nabla_\lambda \text{ELBO}(\lambda_k, \theta; x)$$

4. Update θ based on $\frac{d \text{ELBO}(\lambda_K, \theta; x)}{d\theta}$
5. Update ϕ based on $\frac{d \text{ELBO}(\lambda_K, \theta; x)}{d\phi}$

As in AVI, we make use of a global inference network which outputs variational parameters (step 2). Then, as in SVI, we perform local optimization to refine the

variational parameters for K steps (step 3). Note that the above notation distinguishes between the *gradient* $\nabla_{\theta} f$ and the *total derivative* $\frac{df}{d\theta}$. To be more precise, we use $\nabla_{\mathbf{u}_i} f(\hat{\mathbf{u}}) \in \mathbb{R}^{\dim(\mathbf{u}_i)}$ to refer to the i -th block of the gradient of f evaluated at $\hat{\mathbf{u}} = [\hat{\mathbf{u}}_1, \dots, \hat{\mathbf{u}}_m]$, and further use $\frac{df}{d\mathbf{v}}$ to denote the total derivative of f with respect to \mathbf{v} , which exists if \mathbf{u} is a differentiable function of \mathbf{v} . In general $\nabla_{\mathbf{u}_i} f(\hat{\mathbf{u}}) \neq \frac{df}{d\mathbf{u}_i}$ since other components of \mathbf{u} could be a function of \mathbf{u}_i . This will indeed be the case in our approach; when we calculate $\text{ELBO}(\lambda_K, \theta; x)$, λ_K is a function of the data point x , the generative model θ , and the inference network ϕ .

For training we need to compute the total derivative of the final ELBO with respect to θ, ϕ (i.e., steps 4 and 5 above). Unlike in AVI, in order to update the encoder and generative model parameters, this total derivative requires backpropagating through the SVI updates. Specifically this requires backpropagating through gradient ascent (Domke, 2012; Maclaurin et al., 2015). Following past work, this backpropagation step can be done efficiently with fast Hessian-vector products (LeCun et al., 1993; Pearlmutter, 1994). Consider the case where we perform one step of refinement,

$$\lambda_1 = \lambda_0 + \alpha \nabla_{\lambda} \text{ELBO}(\lambda_0, \theta; x),$$

and for brevity let

$$\mathcal{L} = \text{ELBO}(\lambda_1, \theta; x).$$

To backpropagate through this, we receive the derivative $\frac{d\mathcal{L}}{d\lambda_1}$ and use the chain rule, where $\mathbf{H}_{\mathbf{u}_i, \mathbf{u}_j} f(\hat{\mathbf{u}}) \in \mathbb{R}^{\dim(\mathbf{u}_i) \times \dim(\mathbf{u}_j)}$ is the matrix formed by taking the i -th group of rows and the j -th group of columns of the Hessian of f evaluated at $\hat{\mathbf{u}}$. We can then backpropagate $\frac{d\mathcal{L}}{d\lambda_0}$ through the inference network to calculate the total derivative, i.e.

Algorithm 1 Semi-Amortized Variational Autoencoders

Input: inference network ϕ , generative model θ ,
inference steps K , learning rate α , momentum γ ,
loss function $f(\lambda, \theta, x) = -\text{ELBO}(\lambda, \theta; x)$
Sample $x \sim p_{\mathcal{D}}(x)$
 $\lambda_0 \leftarrow \text{enc}(x; \phi)$
 $v_0 \leftarrow 0$
for $k = 0$ **to** $K - 1$ **do**
 $v_{k+1} \leftarrow \gamma v_k - \nabla_{\lambda} f(\lambda_k, \theta, x)$
 $\lambda_{k+1} \leftarrow \lambda_k + \alpha v_{k+1}$
end for
 $\mathcal{L} \leftarrow f(\lambda_K, \theta, x)$
 $\bar{\lambda}_K \leftarrow \nabla_{\lambda} f(\lambda_K, \theta, x)$
 $\bar{\theta} \leftarrow \nabla_{\theta} f(\lambda_K, \theta, x)$
 $\bar{v}_K \leftarrow 0$
for $k = K - 1$ **to** 0 **do**
 $\bar{v}_{k+1} \leftarrow \bar{v}_{k+1} + \alpha \bar{\lambda}_{k+1}$
 $\bar{\lambda}_k \leftarrow \bar{\lambda}_{k+1} - \text{H}_{\lambda, \lambda} f(\lambda_k, \theta, x) \bar{v}_{k+1}$
 $\bar{\theta} \leftarrow \bar{\theta} - \text{H}_{\theta, \lambda} f(\lambda_k, \theta, x) \bar{v}_{k+1}$
 $\bar{v}_k \leftarrow \gamma \bar{v}_{k+1}$
end for
 $\frac{d\mathcal{L}}{d\bar{\theta}} \leftarrow \bar{\theta}$
 $\frac{d\mathcal{L}}{d\phi} \leftarrow \frac{d\lambda_0}{d\phi} \bar{\lambda}_0$
Update θ, ϕ based on $\frac{d\mathcal{L}}{d\bar{\theta}}, \frac{d\mathcal{L}}{d\phi}$

$\frac{d\mathcal{L}}{d\phi} = \frac{d\lambda_0}{d\phi} \frac{d\mathcal{L}}{d\lambda_0}$. Similar rules can be used to derive $\frac{d\mathcal{L}}{d\bar{\theta}}$.¹ The full forward/backward step, which uses gradient descent with momentum on the negative ELBO, is shown in Algorithm 1.

3.3.1 IMPLEMENTATION DETAILS

In our implementation we calculate Hessian-vector products with finite differences (LeCun et al., 1993; Domke, 2012), which was found to be more memory-efficient than automatic differentiation, and therefore crucial for scaling our approach to rich

¹We refer the reader to Domke (2012) for the full derivation.

inference networks/generative models. Specifically, we estimate $\mathbf{H}_{\mathbf{u}_i, \mathbf{u}_j} f(\hat{\mathbf{u}}) \mathbf{v}$ with

$$\mathbf{H}_{\mathbf{u}_i, \mathbf{u}_j} f(\hat{\mathbf{u}}) \mathbf{v} \approx \frac{1}{\epsilon} \left(\nabla_{\mathbf{u}_i} f(\hat{\mathbf{u}}_0, \dots, \hat{\mathbf{u}}_j + \epsilon \mathbf{v}, \dots, \hat{\mathbf{u}}_m) - \nabla_{\mathbf{u}_i} f(\hat{\mathbf{u}}_0, \dots, \hat{\mathbf{u}}_j, \dots, \hat{\mathbf{u}}_m) \right)$$

where ϵ is some small number (we use $\epsilon = 10^{-5}$).² We further clip the results (i.e. rescale the results if the norm exceeds a threshold) before and after each Hessian-vector product as well as during SVI, which helped mitigate exploding gradients and further gave better training signal to the inference network.

$$\begin{aligned} \frac{d\mathcal{L}}{d\lambda_0} &= \frac{d\lambda_1}{d\lambda_0} \frac{d\mathcal{L}}{d\lambda_1} \\ &= (\mathbf{I} + \alpha \mathbf{H}_{\lambda, \lambda} \text{ELBO}(\lambda_0, \theta; x)) \frac{d\mathcal{L}}{d\lambda_1} \\ &= \frac{d\mathcal{L}}{d\lambda_1} + \alpha \mathbf{H}_{\lambda, \lambda} \text{ELBO}(\lambda_0, \theta; x) \frac{d\mathcal{L}}{d\lambda_1} \end{aligned}$$

Concretely, we define the $\text{clip}(\cdot)$ function as

$$\text{clip}(\mathbf{u}, \eta) = \begin{cases} \frac{\eta}{\|\mathbf{u}\|_2} \mathbf{u}, & \text{if } \|\mathbf{u}\|_2 > \eta \\ \mathbf{u}, & \text{otherwise} \end{cases}$$

and use this to clip the results at various points. Algorithm 2 shows the modified version of Algorithm 1 which makes use of clipping,³ and we use this to perform end-to-

²Since in our case the ELBO is a non-deterministic function due to sampling (and dropout, if applicable), care must be taken when calculating Hessian-vector product with finite differences to ensure that the source of randomness is the same when calculating the two gradient expressions.

³Without gradient clipping, in addition to numerical issues we empirically observed the model to degenerate to a case whereby it learned to rely too much on iterative inference, and thus the initial parameters from the inference network were poor. Another way to provide better signal to the inference network is to train against a weighted sum $\sum_{k=0}^K w_k \text{ELBO}(\lambda_k, \theta; x)$ for $w_k \geq 0$.

Algorithm 2 Semi-Amortized Variational Autoencoders with Clipping

Input: inference network ϕ , generative model θ ,
inference steps K , learning rate α , momentum γ ,
loss function $f(\lambda, \theta, x) = -\text{ELBO}(\lambda, \theta; x)$,
gradient clipping parameter η

Sample $x \sim p_{\mathcal{D}}(x)$
 $\lambda_0 \leftarrow \text{enc}(x; \phi)$
 $v_0 \leftarrow 0$
for $k = 0$ **to** $K - 1$ **do**
 $v_{k+1} \leftarrow \gamma v_k - \text{clip}(\nabla_{\lambda} f(\lambda_k, \theta, x), \eta)$
 $\lambda_{k+1} \leftarrow \lambda_k + \alpha v_{k+1}$
end for
 $\mathcal{L} \leftarrow f(\lambda_K, \theta, x)$
 $\bar{\lambda}_K \leftarrow \nabla_{\lambda} f(\lambda_K, \theta, x)$
 $\bar{\theta} \leftarrow \nabla_{\theta} f(\lambda_K, \theta, x)$
 $\bar{v}_K \leftarrow 0$
for $k = K - 1$ **to** 0 **do**
 $\bar{v}_{k+1} \leftarrow \bar{v}_{k+1} + \alpha \bar{\lambda}_{k+1}$
 $\bar{\lambda}_k \leftarrow \bar{\lambda}_{k+1} - \text{H}_{\lambda, \lambda} f(\lambda_k, \theta, x) \bar{v}_{k+1}$
 $\bar{\lambda}_k \leftarrow \text{clip}(\bar{\lambda}_k, \eta)$
 $\bar{\theta} \leftarrow \bar{\theta} - \text{clip}(\text{H}_{\theta, \lambda} f(\lambda_k, \theta, x) \bar{v}_{k+1}, \eta)$
 $\bar{v}_k \leftarrow \gamma \bar{v}_{k+1}$
end for
 $\frac{d\mathcal{L}}{d\bar{\theta}} \leftarrow \bar{\theta}$
 $\frac{d\mathcal{L}}{d\phi} \leftarrow \frac{d\lambda_0}{d\phi} \bar{\lambda}_0$
Update θ, ϕ based on $\frac{d\mathcal{L}}{d\bar{\theta}}, \frac{d\mathcal{L}}{d\phi}$

end training of our generative model θ and inference network ϕ .

3.4 EMPIRICAL STUDY

3.4.1 EXPERIMENTAL SETUP

DATA We apply our approach to train a generative model of text on the commonly-used Yahoo questions corpus from Yang et al. (2017), which has 100K examples for training and 10K examples for validation/test. Each example consists of a question

followed by an answer from Yahoo Answers. The preprocessed dataset from Yang et al. (2017) takes the top 20K words as the vocabulary after lower-casing all tokens.

HYPERPARAMETERS The architecture and hyperparameters are identical to the LSTM-VAE baselines considered in Yang et al. (2017), except that we train with SGD instead of Adam, which was found to perform better for training LSTMs. Specifically, both the inference network and the generative model are one-layer LSTMs with 1024 hidden units and 512-dimensional word embeddings. The last hidden state of the encoder is used to predict the vector of variational posterior means/log variances, as outlined in 3.2.1. The reparameterized sample from the variational posterior is used to predict the initial hidden state of the generative LSTM and additionally fed as input at each time step. The latent variable is 32-dimensional. Following previous works (Bowman et al., 2016; Sønderby et al., 2016; Yang et al., 2017), we utilize a KL-cost annealing strategy whereby the multiplier on the KL term is increased linearly from 0.1 to 1.0 each batch over 10 epochs. All models are trained with stochastic gradient descent with batch size 32 and learning rate 1.0, where the learning rate starts decaying by a factor of 2 each epoch after the first epoch at which validation performance does not improve. This learning rate decay is not triggered for the first 15 epochs to ensure adequate training. We train for 30 epochs or until the learning rate has decayed 5 times, which was enough for convergence for all models. The model parameters are initialized over $\mathcal{U}(-0.1, 0.1)$ and gradients are clipped at 5. For models trained with iterative inference we perform SVI via stochastic gradient descent with momentum 0.5 and learning rate 1.0. Gradients are clipped during the forward/backward SVI steps, also at 5 (see Algorithm 2).

BASELINES In addition to autoregressive/VAE/SVI baselines, we consider two other approaches that also combine amortized inference with iterative refinement. The first approach is from [Krishnan et al. \(2018\)](#), where the generative model takes a gradient step based on the final variational parameters and the inference network takes a gradient step based on the initial variational parameters, i.e. we update θ based on $\nabla_{\theta} \text{ELBO}(\lambda_K, \theta; x)$ and update ϕ based on $\frac{d\lambda_0}{d\phi} \nabla_{\lambda} \text{ELBO}(\lambda_0, \theta; x)$. The forward step (steps 1-3) is identical to SA-VAE. We refer to this baseline as VAE + SVI. In the second approach, based on [Salakhutdinov & Larochelle \(2010\)](#) and [Hjelm et al. \(2016\)](#), we train the inference network to minimize the KL-divergence between the initial and the final variational distributions, keeping the latter fixed. Specifically, letting $g(\nu, \omega) = \text{KL}[q(\mathbf{z} | x; \nu) \| q(\mathbf{z} | x; \omega)]$, we update θ based on $\nabla_{\theta} \text{ELBO}(\lambda_K, \theta; x)$ and update ϕ based on $\frac{d\lambda_0}{d\phi} \nabla_{\nu} g(\lambda_0, \lambda_K)$. Note that the inference network is not updated based on $\frac{dg}{d\phi}$, which would take into account the fact that both λ_0 and λ_K are functions of ϕ . We found $g(\lambda_0, \lambda_K)$ to perform better than the reverse direction $g(\lambda_K, \lambda_0)$. We refer to this setup as VAE + SVI + KL.

CODE Our code is available at <https://github.com/harvardnlp/sa-vae>.

3.4.2 RESULTS

Results from the various models are shown in Table 3.1. Our baseline models (LM/VAE/SVI in Table 3.1) are already quite strong, however models trained with VAE/SVI make negligible use of the latent variable and practically collapse to a language model, as first observed by [Bowman et al. \(2016\)](#).⁴ In contrast, models that combine amortized

⁴Models trained with word dropout (+ WORD-DROP in Table 3.1) do make use of the latent space but significantly underperform a language model.

MODEL	NLL	KL	PPL
LSTM-LM	334.9	–	66.2
LSTM-VAE	≤ 342.1	0.0	≤ 72.5
LSTM-VAE + INIT	≤ 339.2	0.0	≤ 69.9
CNN-LM	335.4	–	66.6
CNN-VAE	≤ 333.9	6.7	≤ 65.4
CNN-VAE + INIT	≤ 332.1	10.0	≤ 63.9
LM	329.1	–	61.6
VAE	≤ 330.2	0.01	≤ 62.5
VAE + INIT	≤ 330.5	0.37	≤ 62.7
VAE + WORD-DROP 25%	≤ 334.2	1.44	≤ 65.6
VAE + WORD-DROP 50%	≤ 345.0	5.29	≤ 75.2
SVI ($K = 10$)	≤ 331.4	0.16	≤ 63.4
SVI ($K = 20$)	≤ 330.8	0.41	≤ 62.9
SVI ($K = 40$)	≤ 329.8	1.01	≤ 62.2
VAE + SVI ($K = 10$)	≤ 331.2	7.85	≤ 63.3
VAE + SVI ($K = 20$)	≤ 330.5	7.80	≤ 62.7
VAE + SVI + KL ($K = 10$)	≤ 330.3	7.95	≤ 62.5
VAE + SVI + KL ($K = 20$)	≤ 330.1	7.81	≤ 62.3
SA-VAE ($K = 10$)	≤ 327.6	5.13	≤ 60.5
SA-VAE ($K = 20$)	≤ 327.5	7.19	≤ 60.4

Table 3.1: Results on the Yahoo dataset. Top results are from [Yang et al. \(2017\)](#), while the bottom results are from this work. For latent variable models we show the negative ELBO which upper bounds the negative log likelihood (NLL). Models with + INIT means the encoder is initialized with a pretrained language model, while models with + WORD-DROP are trained with word-dropout. KL portion of the ELBO indicates latent variable usage, and PPL refers to perplexity. K refers to the number of inference steps used for training/testing.

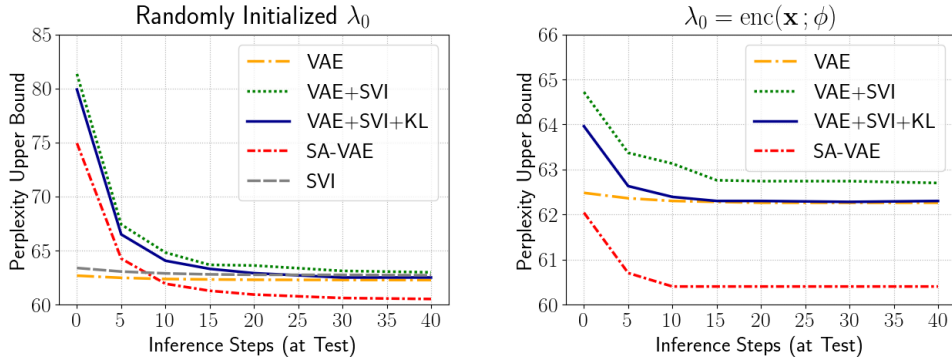


Figure 3.1: (Left) Perplexity upper bound of various models when trained with 20 steps (except for VAE) and tested with varying number of SVI steps from random initialization. (Right) Same as the left except that SVI is initialized with variational parameters obtained from the inference network.

inference with iterative refinement make use of the latent space and the KL term is significantly above zero.⁵ VAE + SVI and VAE + SVI + KL do not outperform a language model, while SA-VAE modestly outperforms it.

One might wonder if the improvements are coming from simply having a more flexible inference scheme at test time, rather than from learning a better generative model. To test this, for the various models we discard the inference network at test time and perform SVI for a variable number of steps from random initialization. The results are shown in Figure 3.1 (left). It is clear that the learned generative model (and the associated ELBO landscape) is quite different—it is not possible to train with VAE and perform SVI at test time to obtain the same performance as SA-VAE (although the performance of VAE does improve slightly from 62.7 to 62.3 when we run SVI for 40 steps from random initialization). Figure 3.1 (right) has the results for a similar experiment where we refine the variational parameters initialized from

⁵A high KL term does not necessarily imply that the latent variable is being utilized in a meaningful way (it could simply be due to bad optimization). In later sections we investigate the learned latent space in more detail.

the inference network for a variable number of steps at test time. We find that the inference network provides better initial parameters than random initialization and thus requires fewer iterations of SVI to reach the optimum. We do not observe improvements for running more refinement steps than was used in training at test time. Interestingly, SA-VAE without any refinement steps at test time has a substantially nonzero KL term (KL = 6.65, PPL = 62.0). This indicates that the posterior collapse phenomenon when training LSTM-based VAEs for text is partially due to optimization issues. Finally, while Yang et al. (2017) found that initializing the encoder with a pretrained language model improved performance (+ INIT in Table 3.1), we did not observe this on our baseline VAE model when we trained with SGD and hence did not pursue this further.

3.4.3 ANALYSIS OF LATENT VARIABLES

We investigate what the latent variables are learning through saliency analysis with our best model (SA-VAE trained with 20 steps). Specifically, we calculate the output saliency of each token x_t with respect to \mathbf{z} as

$$\mathbb{E}_{q(\mathbf{z};\lambda)} \left[\left\| \frac{d \log p(x_t | x_{<t}, \mathbf{z}; \theta)}{d\mathbf{z}} \right\|_2 \right],$$

where $\|\cdot\|_2$ is the l_2 norm and the expectation is approximated with 5 samples from the variational posterior. Saliency is therefore a measure of how much the latent variable is being used to predict a particular token. We visualize the output saliency of a few examples from the test set in Figure 3.2 (top).

The previous definition of saliency measures the influence of \mathbf{z} on the *output* x_t . We can also roughly measure the influence of the *input* x_t on the latent representation \mathbf{z} ,

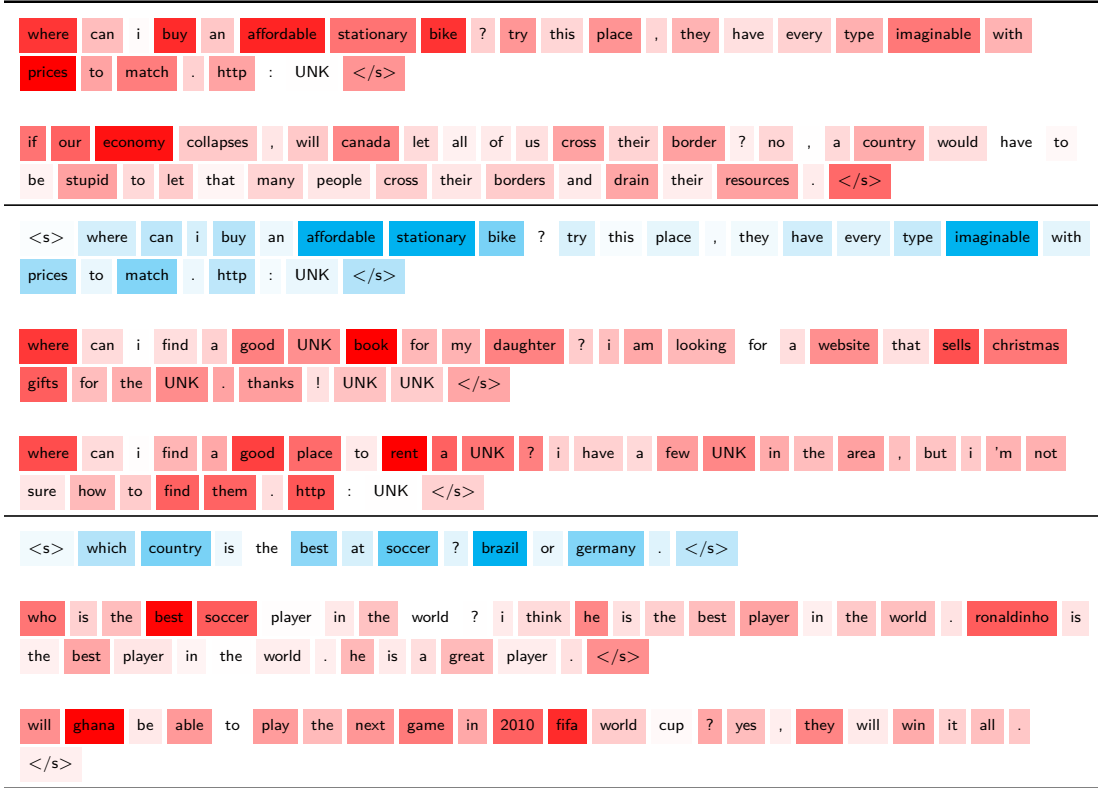


Figure 3.2: (Top) Output saliency visualization of some examples from the test set. Here the saliency values are rescaled to be between 0-100 within each example for easier visualization. Red indicates higher saliency values. (Middle) Input saliency of the first test example from the top (in blue), in addition to two sample outputs generated from the variational posterior (with their saliency values in red). (Bottom) Same as the middle except we use a made-up example..

which we refer to as input saliency:

$$\left\| \mathbb{E}_{q(\mathbf{z}; \lambda)} \left[\frac{d \|\mathbf{z}\|_2}{d \mathbf{w}_t} \right] \right\|_2$$

Here \mathbf{w}_t is the encoder word embedding for x_t .⁶ We visualize the input saliency for a test example (Figure 3.2, middle) and a made-up example (Figure 3.2, bottom).

⁶As the norm of \mathbf{z} is a rather crude measure, a better measure would be obtained by analyzing the spectra of the Jacobian $\frac{d\mathbf{z}}{d\mathbf{w}_t}$. However this is computationally too expensive to calculate for each token in the corpus.

Under each input example we also visualize a two samples from the variational posterior, and find that the generated examples are often meaningfully related to the input example.⁷ From a qualitative analysis of saliency, several things are apparent: the latent variable seems to encode question type (i.e. if, what, how, why, etc.) and therefore saliency is high for the first word; content words (nouns, adjectives, lexical verbs) have much higher saliency than function words (determiners, prepositions, conjunctions, etc.); saliency of the $\langle /s \rangle$ token is quite high, indicating that the length information is also encoded in the latent space. In the third example we observe that the left parenthesis has higher saliency than the right parenthesis (0.32 vs. 0.24 on average across the test set), as the latter can be predicted by conditioning on the former rather than on the latent representation \mathbf{z} .

We quantitatively analyze output/input saliency across part-of-speech, token position, and word frequency in Figure 3.3: nouns (NN), adjectives (JJ), verbs (VB), numbers (CD), and the $\langle /s \rangle$ token have higher saliency than conjunctions (CC), determiners (DT), prepositions (IN), and the TO token—the latter are relatively easier to predict by conditioning on previous tokens; similarly, on average, tokens occurring earlier have much higher saliency than those occurring later (Figure 3.3 shows absolute position but the plot is similar with relative position); the latent variable is used much more when predicting rare tokens. These results seem to suggest that the latent variables are encoding interesting and potentially interpretable aspects of language. While left as future work, it is possible that manipulations in the latent space of a model learned this way could lead to controlled generation/manipulation of output text (Hu et al., 2017; Mueller et al., 2017).

⁷We first sample $\mathbf{z} \sim q(\mathbf{z} | x; \lambda_K)$ then $x \sim p(x | \mathbf{z}; \theta)$. When sampling $x_t \sim p(x_t | x_{<t}, \mathbf{z}; \theta)$ we sample with temperature $T = 0.25$, i.e. $p(x_t | x_{<t}, \mathbf{z}; \theta) = \text{softmax}(\frac{1}{T} \mathbf{s}_{t-1})$ where \mathbf{s}_t is the vector with scores for all words.

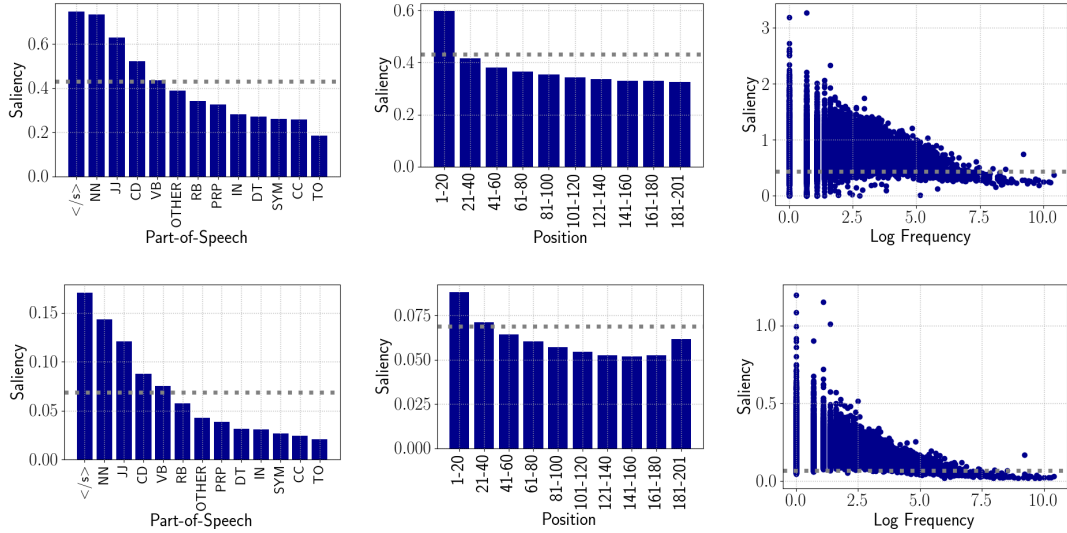


Figure 3.3: Saliency by part-of-speech tag, position, and log frequency for the output (top) and the input (bottom). See text for the definitions of input/output saliency. The dotted gray line in each plot shows the average saliency across all words.

3.5 DISCUSSION

3.5.1 LIMITATIONS

A drawback of our approach (and other non-amortized inference methods) is that each training step requires backpropagating through the generative model multiple times, which can be costly especially if the generative model is expensive to compute. This may potentially be mitigated through more sophisticated meta learning approaches (Andrychowicz et al., 2016; Marino et al., 2018), or with more efficient use of the past gradient information during SVI via averaging (Schmidt et al., 2013) or importance sampling (Sakaya & Klami, 2017). One could also consider employing synthetic gradients (Jaderberg et al., 2017) to limit the number of backpropagation steps during training. Krishnan et al. (2018) observe that it is more important to

train with iterative refinement during earlier stages (we also observed this in preliminary experiments), and therefore annealing the number of refinement steps as training progresses could also speed up training.

Our approach is mainly applicable to variational families that avail themselves to differentiable optimization (e.g. gradient ascent) with respect to the ELBO. In contrast, VAE + SVI and VAE + SVI + KL are applicable to more general optimization algorithms.

3.5.2 POSTERIOR COLLAPSE: OPTIMIZATION VS. UNDERLYING MODEL

From the perspective of learning meaningful, interesting, and useful latent variables, one might question the prudence of using an autoregressive model that fully conditions on its entire history (as opposed to assuming some conditional independence) given that $p(x)$ can always be factorized as $\prod_{t=1}^T p(x_t | x_{<t})$, and therefore the model is *non-identifiable*. We saw in 3.2.2 the generative model does not have to utilize the latent variable if it is powerful enough.⁸ Our results however indicate that posterior collapse is at least partially caused by optimization issues, and can be mitigated through better optimization of the variational distribution. Subsequent work has found that improving optimization dynamics during training through various strategies, for example (1) updating the inference network more aggressively (He et al., 2019), (2) utilizing a cyclic scheduling of penalty on the KL term (Fu et al., 2019), and (3) using a modified objective that does not penalize KL-usage up to a certain threshold (Li et al., 2019b), all allow for the use of fully autoregressive generative models that do not ignore the latent variable.

⁸Indeed, such an argument has been used to favor generative models that exhibit independence properties conditioned on \mathbf{z} (Chen et al., 2017b; Miao et al., 2017; Yang et al., 2017), and in chapter 6 we experiment further with such *conditionally Markov* models.

3.6 RELATED WORK

The semi-amortized approach introduced in this chapter is closely related to the line of work which uses a separate model to initialize variational parameters and subsequently updates them through an iterative procedure (Salakhutdinov & Larochelle, 2010; Cho et al., 2013; Salimans et al., 2015; Hjelm et al., 2016; Krishnan et al., 2018; Pu et al., 2017). Marino et al. (2018) utilize meta-learning to train an inference network which learns to perform iterative inference by training a deep model to output the variational parameters for each time step.

While differentiating through inference/optimization was initially explored by various researchers primarily outside the area of deep learning (Stoyanov et al., 2011; Domke, 2012; Brakel et al., 2013), they have more recently been explored in the context of hyperparameter optimization (Maclaurin et al., 2015) and as a differentiable layer of a deep model (Belanger et al., 2017; Kim et al., 2017; Metz et al., 2017; Amos & Kolter, 2017).

Initial work on VAE-based approaches to image modeling focused on simple generative models that assumed independence among pixels conditioned on the latent variable (Kingma & Welling, 2014; Rezende et al., 2014). More recent works have obtained substantial improvements in log-likelihood and sample quality through utilizing powerful autoregressive models (PixelCNN) as the generative model (Chen et al., 2017b; Gulrajani et al., 2017). In contrast, modeling text with VAEs has remained challenging. Bowman et al. (2016) found that using an LSTM generative model resulted in a degenerate case whereby the variational posterior collapsed to the prior and the generative model ignored the latent code (even with richer variational families). Many works on VAEs for text have thus made simplifying conditional inde-

pendence assumptions (Miao et al., 2016), used less powerful generative models such as convolutional networks (Yang et al., 2017; Semeniuta et al., 2017), or combined a recurrent generative model with a topic model (Dieng et al., 2017; Wang et al., 2018a). Note that unlike to sequential VAEs that employ different latent variables at each time step (Chung et al., 2015; Fraccaro et al., 2016; Krishnan et al., 2017; Serban et al., 2017; Goyal et al., 2017a), in this work we focus on modeling the entire sequence with a global latent variable.

Finally, since our work only addresses the *amortization gap* (the gap between the log-likelihood and the ELBO due to amortization) and not the *approximation gap* (due to the choice of a particular variational family) (Cremer et al., 2018), it can be combined with existing work on employing richer posterior/prior distributions within the VAE framework (Rezende & Mohamed, 2015a; Kingma et al., 2016; Johnson et al., 2016; Tran et al., 2016; Goyal et al., 2017b; Guu et al., 2017; Tomczak & Welling, 2018).

3.7 CONCLUSION

In this chapter we have considered a continuous latent variable model of text where each sentence/paragraph is generated from a continuous vector. We introduced an improved, semi-amortized approach for training such models, which combines amortized variational inference with stochastic variational inference. With the approach we found that we were able to train deep latent variable models of text with an expressive autogressive generative model that does not ignore the latent variable. Training generative models that both model the underlying data well and learn useful latent representations is an important avenue for future work.

4

Latent Variable Model of Attention & Relaxations of Discrete Spaces

4.1 INTRODUCTION

In this chapter we consider a discrete latent variable model of attention within the sequence-to-sequence learning framework (Bahdanau et al., 2015). We model *latent*

The material in this chapter is adapted from Deng et al. (2018).

alignment between source/target sentences as a sequence of discrete latent variables in the context of neural machine translation (MT) systems.

Learning latent word alignments from parallel sentences has long been a core problem in statistical MT and represents one of the most successful applications of latent variable modeling in NLP, starting with the seminal IBM models (Brown et al., 1993), HMM-based alignment models (Vogel et al., 1996), and a fast log-linear reparameterization of the IBM 2 model (Dyer et al., 2013). In contrast to statistical MT systems, neural MT systems directly model the full distribution over the target sentence conditioned on the source sentence without explicitly modeling alignments as an intermediate step. They instead makes use of *soft alignments* via the soft attention mechanism (Bahdanau et al., 2015). Alongside components such as residual blocks and batch/layer normalization, soft attention provides a rich neural network building block for controlling gradient flow and encoding inductive biases, and are now part of the standard deep learning toolkit. However, more so than these other components, which are generally treated as black boxes, researchers often use intermediate attention decisions directly as a tool for model interpretability (Lei et al., 2016; Alvarez-Melis & Jaakkola, 2017a) or as a factor in final predictions (Shin et al., 2017). From this perspective, soft attention plays the role of implicitly approximating latent alignments as in statistical MT systems (Brown et al., 1993; Koehn et al., 2007).

While soft attention works well empirically, explicitly modeling alignment as a latent variable remains appealing for several reasons: (1) latent variables facilitate reasoning about dependencies in a modular and probabilistically principled way, e.g. allowing composition with other models, (2) posterior inference provides a better basis for model analysis and partial predictions than strictly feed-forward models, which have been shown to underperform on alignment in machine translation (Koehn &

Knowles, 2017), and finally (3) latent variable modeling may lead to better predictive accuracy when properly trained. However, straightforwardly modeling attention at each time step as a categorical latent variable (*hard* or *discrete* attention) has been known to underperform (deterministic) soft attention, aside from a few exceptions (Xu et al., 2015). In this chapter we revisit discrete attention models and show that their underperformance vis-à-vis soft attention is due to a large gap between the log marginal likelihood and the lower bound objective optimized by previous discrete attention models. We propose an efficient *variational attention* approach that closes this gap by training an inference network over the source and target sentence which approximates the posterior attention distribution. Optimizing the evidence lower bound with discrete latent variables is challenging since the reparameterization trick described in 2.5.5 is not directly applicable. We experiment with two approaches to obtaining low-variance estimators: (1) a score function gradient estimator with a variance-reducing baseline from an auxiliary soft attention model and, (2) a pathwise gradient estimator using the Gumbel-Softmax distribution (Jang et al., 2017; Maddison et al., 2017) which relaxes the discrete space into a continuous space. We find that with these approaches, we are able to learn discrete attention models which outperform their soft counterparts and at the same time allow for greater model introspection through posterior inference.

4.2 BACKGROUND

We briefly review the attention mechanism in neural sequence-to-sequence learning, in particular soft attention (4.2.1) and hard/discrete attention (4.2.2). We also discuss the Gumbel-Softmax distribution for approximately obtaining pathwise gradient

estimators for discrete distributions (4.2.3).

4.2.1 SOFT ATTENTION

In sequence-to-sequence learning with attention networks (Bahdanau et al., 2015), we model the distribution over the target sentence $y = [y_1, \dots, y_T]$ given the source sentence $x = [x_1, \dots, x_S]$ with an autoregressive model,

$$p(y | x; \theta) = \prod_{t=1}^T p(y_t | x, y_{<t}; \theta),$$

where the distribution over the next token is given by an affine transformation followed by a softmax over a feature vector \mathbf{c}_t ,

$$p(y_t | x, y_{<t}; \theta) = \text{softmax}(\mathbf{W}\mathbf{c}_t + \mathbf{b})[y_t].$$

The vector \mathbf{c}_t combines the previous *decoder* hidden state with an attention over *encoder* hidden states.¹ Concretely, we first run sequential neural networks over the source and target to obtain contextualized vector representations of each word,

$$\mathbf{s}_{1:S} = \text{SeqNN}(\mathbf{x}_{1:S}; \theta_{\text{encoder}}),$$

$$\mathbf{h}_{1:T} = \text{SeqNN}(\mathbf{y}_{1:T}; \theta_{\text{decoder}}),$$

¹Note that our use of “encoder” here refers to the sequential neural network over the source sentence, and not the inference network in variational autoencoders.

where $\mathbf{x}_{1:S}$ and $\mathbf{y}_{1:T}$ are the source/target word embeddings. Then, we obtain an attention distribution over the source hidden states via a softmax,

$$\alpha_{t,i} = \frac{\exp(f(\mathbf{s}_i, \mathbf{h}_{t-1}; \theta_{\text{attention}}))}{\sum_{j=1}^S \exp(f(\mathbf{s}_j, \mathbf{h}_{t-1}; \theta_{\text{attention}}))}$$

where $f : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ computes an affinity between the target word y_t and the source word x_i (e.g. a dot product $\mathbf{s}_i^\top \mathbf{h}_{t-1}$ in the simplest case, in which case $\theta_{\text{attention}} = \emptyset$).

We subsequently use the attention distribution to obtain a convex combination of source hidden states, which is finally combined with \mathbf{h}_{t-1} to produce \mathbf{c}_t ,

$$\begin{aligned} \hat{\mathbf{s}}_t &= \sum_{i=1}^S \alpha_{t,i} \mathbf{s}_i \\ \mathbf{c}_t &= \text{MLP}([\mathbf{h}_{t-1}; \hat{\mathbf{s}}_t]; \theta_{\text{context}}). \end{aligned}$$

We refer to the combination of source hidden states $\hat{\mathbf{s}}_t$ as the *context vector* since it aggregates the context source sentence at each time step t . In this encoder-decoder model the parameters are given by

$$\theta = \{\theta_{\text{encoder}}, \theta_{\text{decoder}}, \theta_{\text{attention}}, \theta_{\text{context}}, \mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y}\},$$

where \mathbf{X}/\mathbf{Y} are the source/target word embeddings.

In practice many variants exist, including different parameterizations of SeqNN (e.g. LSTMs (Luong et al., 2015), GRUs (Bahdanau et al., 2015), and Transformers (Vaswani et al., 2017)), and the attention scoring function $f(\cdot)$ (e.g. MLP attention (Bahdanau et al., 2015) and bilinear attention (Luong et al., 2015)). Empirically the soft attention mechanism works remarkably well, and alignment distributions learned

this way often correspond to intuition (e.g. word alignment in machine translation).

4.2.2 HARD ATTENTION

Note that there are no latent variables in the soft attention mechanism; it is a deterministic feed-forward function. In contrast, in *hard* or *discrete* attention we posit a categorical latent variable $z_t \in \{1, \dots, S\}$ at each time step whose distribution is given by the attention scores, i.e.

$$p(z_t = i \mid x, y_{<t}; \theta) = \alpha_{t,i} = \frac{\exp(f(\mathbf{s}_i, \mathbf{h}_{t-1}; \theta_{\text{attention}}))}{\sum_{j=1}^S \exp(f(\mathbf{s}_j, \mathbf{h}_{t-1}; \theta_{\text{attention}}))}.$$

Under this formulation, the distribution over the target token at each time step is given by marginalizing over the unobserved latent variable z_t ,

$$\begin{aligned} \log p(y_t \mid y_{<t}, x; \theta) &= \log \left(\sum_{i=1}^S p(z_t = i \mid x, y_{<t}; \theta) p(y_t \mid y_{<t}, x, z; \theta) \right) \\ &= \log \left(\sum_{i=1}^S \alpha_{t,i} p(y_t \mid y_{<t}, x, z; \theta) \right), \end{aligned}$$

where

$$\begin{aligned} p(y_t \mid y_{<t}, x, z; \theta) &= \text{softmax}(\mathbf{W} \text{MLP}([\mathbf{h}_t; \bar{\mathbf{s}}_t]) + \mathbf{b})[y_t], \\ \bar{\mathbf{s}}_t &= \sum_{i=1}^S \mathbb{1}\{z_t = i\} \mathbf{s}_i \end{aligned}$$

That is, the context vector $\bar{\mathbf{s}}_t$ in hard attention is obtained by a hard selection of a vector in $\mathbf{s}_{1:S}$, instead of a convex combination as in soft attention. Another way to

illuminate the difference between the two is in terms of an expectation over z_t ,

$$\textbf{Hard Attention} : \log p(y_t | y_{<t}, x; \theta) = \log (\mathbb{E} [\text{softmax} (\mathbf{W} \text{MLP} ([\mathbf{h}_t; \bar{\mathbf{s}}_t]) + \mathbf{b})]) ,$$

$$\textbf{Soft Attention} : \log p(y_t | y_{<t}, x; \theta) = \log (\text{softmax} (\mathbf{W} \text{MLP} ([\mathbf{h}_t; \mathbb{E}[\bar{\mathbf{s}}_t]]) + \mathbf{b})) ,$$

which holds since

$$\begin{aligned} \mathbb{E}_{p(z_t | x, y_{<t}; \theta)} [\bar{\mathbf{s}}_t] &= \sum_{i=1}^S p(z_t = i | x, y_{<t}; \theta) \mathbf{s}_i \\ &= \hat{\mathbf{s}}_t. \end{aligned}$$

That is, whereas hard attention has an expectation *after* the softmax function over the vocabulary, soft attention has an expectation *before* the softmax. This makes hard attention $O(S)$ times more expensive than soft attention, since the softmax must be run S times for each possible value of z_t .²

In cases where there are multiple attention steps (e.g. as in multi-hop attention) or the source sentence length S is large, calculating the log marginal likelihood exactly via enumeration over z_t can be prohibitively expensive. In this case, a common strategy is to maximize a lower bound on the log marginal likelihood using Jensen’s inequality,

$$\begin{aligned} \log p(y_t | y_{<t}, x; \theta) &= \log (\mathbb{E} [\text{softmax} (\mathbf{W} \text{MLP} ([\mathbf{h}_t; \bar{\mathbf{s}}_t]) + \mathbf{b})] [y_t]) \\ &\geq \mathbb{E} [\log(\text{softmax}(\mathbf{W} \text{MLP}([\mathbf{h}_t; \bar{\mathbf{s}}_t] + \mathbf{b})[y_t])] , \end{aligned}$$

²However, the computation before the attention layer to get $\mathbf{s}_{1:S}$ and $\mathbf{h}_{1:T}$ is the same in both hard/soft attention. Also note that the both soft and hard attention models have the same set of parameters θ .

which holds since \log is concave. Then we can perform gradient-based optimization with the score function estimator

$$\begin{aligned} \nabla_{\theta} \mathbb{E}[\log(\text{softmax}(\mathbf{W} \text{MLP}([\mathbf{h}_t; \bar{\mathbf{s}}_t] + \mathbf{b}))[y_t])] &= \mathbb{E} \left[\nabla_{\theta} \log(\text{softmax}(\mathbf{W} \text{MLP}([\mathbf{h}_t; \bar{\mathbf{s}}_t] + \mathbf{b}))[y_t]) \right. \\ &\quad \left. + \log(\text{softmax}(\mathbf{W} \text{MLP}([\mathbf{h}_t; \bar{\mathbf{s}}_t] + \mathbf{b}))[y_t]) \times \nabla_{\theta} \log p(z_t | x, y_{<t}; \theta) \right], \end{aligned}$$

where the resulting expectation can be estimated with Monte Carlo samples. This approach to training hard attention models was originally proposed by [Xu et al. \(2015\)](#) for image captioning, who utilized variance reducing control variates derived from a moving average of previous rewards. However, hard attention has generally underperformed soft attention for other tasks ([Ling & Rush, 2017](#)). In section 4.3 we investigate this issue in more detail and propose a variational approach to training hard attention models.

4.2.3 PATHWISE GRADIENT ESTIMATORS FOR DISCRETE DISTRIBUTIONS

We now discuss an alternative to the score function gradient estimator for training discrete latent variable models, which utilizes the *Gumbel-Max trick* ([Papandreou & Yuille, 2011](#); [Hazan & Jaakkola, 2012](#); [Maddison et al., 2014](#)). Concretely, suppose \mathbf{z} is a one-hot representation of a categorical random variable z with K categories and unnormalized scores $\boldsymbol{\alpha}$, i.e.

$$p(z = k; \boldsymbol{\alpha}) = p(\mathbf{z}[k] = 1; \boldsymbol{\alpha}) = \frac{\boldsymbol{\alpha}[k]}{\sum_{i=1}^K \boldsymbol{\alpha}[i]}.$$

Then we can draw a sample from $p(\mathbf{z}; \boldsymbol{\alpha})$ by solving the following optimization problem,

$$\hat{\mathbf{z}} = \arg \max_{\mathbf{t} \in \Delta^{K-1}} (\log \boldsymbol{\alpha} + \mathbf{g})^\top \mathbf{t},$$

where Δ^{K-1} is the $K-1$ -simplex and $\mathbf{g} = [\mathbf{g}[1], \dots, \mathbf{g}[K]]$ is the vector of i.i.d samples from a Gumbel distribution,

$$\mathbf{g}[i] \stackrel{\text{i.i.d.}}{\sim} \text{Gumbel}(0, 1).$$

In this case we have $\hat{\mathbf{z}} \sim p(\mathbf{z}; \boldsymbol{\alpha})$. Thus, the Gumbel-Max trick transforms the problem of sampling from a discrete distribution to an optimization problem of finding the argmax of perturbed logits of a discrete distribution. While this shows that we can reparameterize a discrete distribution, we cannot straightforwardly perform gradient-based optimization using this trick because the arg max function has zero gradients almost everywhere.

If we replace the arg max above with a softmax and a temperature term $\tau > 0$,

$$\begin{aligned} \mathbf{u} &= \text{softmax} \left(\frac{\log \boldsymbol{\alpha} + \mathbf{g}}{\tau} \right), \\ \mathbf{u}[k] &= \frac{\exp((\log \boldsymbol{\alpha}[k] + \mathbf{g}[k])/\tau)}{\sum_{i=1}^K \exp((\log \boldsymbol{\alpha}[i] + \mathbf{g}[i])/\tau)}, \end{aligned}$$

we say that \mathbf{u} is drawn from a *Gumbel-Softmax* distribution (Jang et al., 2017) with parameters $\boldsymbol{\alpha}$ and τ ,³

$$\mathbf{u} \sim \text{Gumbel-Softmax}(\mathbf{u}; \boldsymbol{\alpha}, \tau).$$

The Gumbel-Softmax defines a distribution on the simplex Δ^{K-1} , and the density is

³Contemporaneous work by Maddison et al. (2017) calls this a *Concrete* distribution.

given by

$$p(\mathbf{u}; \boldsymbol{\alpha}, \tau) = (K-1)! \tau^{K-1} \prod_{k=1}^K \left(\frac{\boldsymbol{\alpha}[k] \mathbf{u}[k]^{-\tau-1}}{\sum_{j=1}^K \boldsymbol{\alpha}[j] \mathbf{u}[j]^{-\tau}} \right).$$

(See Jang et al. (2017) and Maddison et al. (2017) for the derivation). Notably this distribution is reparameterizable by construction since we can draw a sample by (1) drawing Gumbel noise \mathbf{g} , (2) transforming the noise as $(\mathbf{g}[i] + \log \boldsymbol{\alpha}[i])/\tau$, and (3) applying a softmax to the transformed values. Importantly, unlike the arg max function, softmax has nonzero gradients. While \mathbf{u} is no longer discrete, we can anneal the temperature $\tau \rightarrow 0$ as training progresses and hope that this will approximate a sample from a discrete distribution.

We now discuss how we can use this to train a variational autoencoder with discrete latent variables. Recall that the gradient of ELBO with respect to ϕ is

$$\begin{aligned} \nabla_{\phi} \text{ELBO}(\theta, \phi; x) &= \nabla_{\phi} \mathbb{E}_{q(\mathbf{z} | x; \phi)} \left[\log \frac{p(x, \mathbf{z}; \theta)}{q(\mathbf{z} | x; \phi)} \right] \\ &= \nabla_{\phi} \mathbb{E}_{q(\mathbf{z} | x; \phi)} [\log p(x | \mathbf{z}; \theta)] - \nabla_{\phi} \text{KL}[q(\mathbf{z} | x; \phi) \| p(\mathbf{z}; \theta)]. \end{aligned}$$

As in the Gaussian case from chapter 3, the KL-divergence between two discrete distributions is easy to calculate and thus $\nabla_{\phi} \text{KL}[q(\mathbf{z} | x; \phi) \| p(\mathbf{z}; \theta)]$ is not an issue.

For the first term, we use the Gumbel-Softmax approximation to the discrete distribution. Suppose now $\boldsymbol{\alpha} = \text{enc}(x; \phi)$ and let $q_{\text{relax}}(\mathbf{u} | x; \phi, \tau)$ be the variational Gumbel-Softmax distribution (note that τ could in theory also be a function of x), where we have used subscripts in $q_{\text{relax}}(\cdot)$ to distinguish it from the original (discrete) variational distribution $q(\mathbf{z} | x; \phi)$. We might then hope that the gradient of the ELBO obtained from the relaxed, Gumbel-Softmax distribution will well-approximate the

true gradient,

$$\begin{aligned}\nabla_{\phi} \mathbb{E}_{q(\mathbf{z} | x; \phi)} [\log p(x | \mathbf{z}; \theta)] &\approx \nabla_{\phi} \mathbb{E}_{q_{\text{relaxed}}(\mathbf{u} | x; \phi, \tau)} [\log p(x | \mathbf{z} = \mathbf{u}; \theta)] \\ &= \mathbb{E}_{\mathbf{g} \sim \text{Gumbel}} [\nabla_{\phi} \log p(x | \mathbf{z} = \text{softmax}((\mathbf{g} + \log \boldsymbol{\alpha})/\tau); \theta)],\end{aligned}$$

where we can now push the gradient sign inside the expectation and use the path-wise gradient estimator in the second inequality.⁴ Unlike the score function gradient estimator, the above estimator is biased and the variance will diverge to infinity as $\tau \rightarrow 0$. Also note the above relaxation is only applicable if the likelihood model $p(x | \mathbf{z}; \theta)$, which was originally defined over a *discrete* latent space $\mathbf{z} \in \{0, 1\}^K$ (and the observed data), is well-defined for a *continuous* latent space $\mathbf{u} \in \Delta^{K-1}$. While this will indeed be the case for hard attention, for many models of interest this kind of relaxation is not possible, for example in models with dynamic computation graphs that depend on different choices of the latent variable.

4.3 VARIATIONAL ATTENTION FOR LATENT ALIGNMENT

Recall that log likelihood over the target sentence in the hard attention model is given by a summation of individual log marginal likelihoods at each time step,

$$\sum_{t=1}^T \log p(y_t | y_{<t}, x; \theta) = \sum_{t=1}^T \log \left(\sum_{i=1}^S p(z_t = i | x, y_{<t}; \theta) p(y_t | y_{<t}, x, z; \theta) \right).$$

⁴Other variants of this approach are possible. For example we could use a straight-through estimator, which use $\arg \max$ on the forward pass and softmax only in the backward pass.

Let us now consider the evidence lower bound for the log marginal likelihood in the hard attention model for a single time step,

$$\begin{aligned} \log p(y_t | y_{<t}, x; \theta) &\geq \mathbb{E}_{q(z_t | x, y; \phi)} [\log p(y_t | x, y_{<t}, z; \theta)] - \\ &\quad \text{KL}[q(z_t | x, y; \phi) \| p(z_t | x, y_{<t}; \theta)]. \end{aligned}$$

We first observe that that the lower bound objective deriving from Jensen’s inequality in section 4.2.2,

$$\mathbb{E}_{p(z_t | x, y_{<t}; \theta)} [\log p(y_t | x, y_{<t}, z_t; \theta)],$$

is equal to the evidence lower bound when we take the variational distribution to be the prior, i.e.,

$$q(z_t | x, y; \phi) = p(z_t | x, y_{<t}; \theta).$$

This choice of variational distribution implicitly assumes that the posterior distribution over z_t is independent from the prior, which is clearly suboptimal when modeling alignment, since knowing y_t can give a strong signal to which word in the source was translated. This can lead to a large gap between the log marginal likelihood.

In this section, we consider a variational approach to minimize this gap by employing an inference network over the source and target sentences, as shown in Figure 4.1. One way to parameterize the inference network is

$$q(z_t = i | x, y; \phi) = \frac{\exp(f(\mathbf{s}_i, \mathbf{h}_t; \phi))}{\sum_{j=1}^S \exp(f(\mathbf{s}_j, \mathbf{h}_t; \phi))}.$$

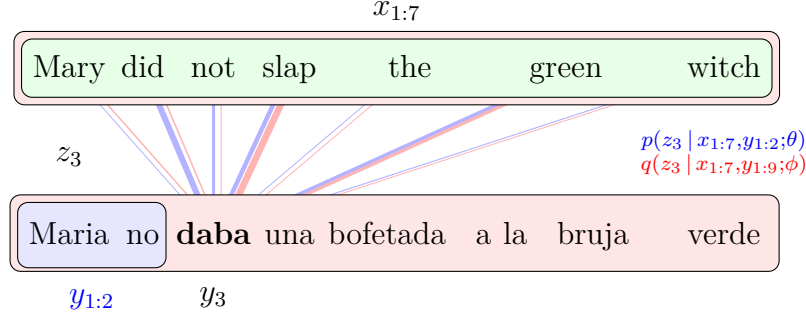


Figure 4.1: Sketch of variational attention applied to machine translation. Here the source sentence $x_{1:7}$ has 7 words and the target sentence $y_{1:9}$ has 9 words. Two alignment distributions are shown, for the blue prior p , and the red variational posterior q taking into account future observations. Our aim is to use q , which conditions on the entire target sentence, to improve estimates of p and to support improved inference of z .

Observe that in contrast to the prior attention distribution which utilizes the previous time’s hidden state \mathbf{h}_{t-1} (because it has not generated y_t yet), the inference network instead uses \mathbf{h}_t , because it has access to all of y , i.e.

$$\begin{aligned} \textbf{Prior Attention : } p(z_t = i | x, y_{<t}; \theta) &= \frac{\exp(f(\mathbf{s}_i, \mathbf{h}_{t-1}; \theta))}{\sum_{j=1}^S \exp(f(\mathbf{s}_j, \mathbf{h}_{t-1}; \theta))}, \\ \textbf{Variational Posterior : } q(z_t = i | x, y; \phi) &= \frac{\exp(f(\mathbf{s}_i, \mathbf{h}_t; \phi))}{\sum_{j=1}^S \exp(f(\mathbf{s}_j, \mathbf{h}_t; \phi))}. \end{aligned}$$

Here we use $\mathbf{s}_i, \mathbf{h}_t$ to refer to the source/target hidden states in both the prior and the variational posterior to emphasize their similarity. However, in practice the inference network has its own SeqNN parameters, and additionally utilizes a bi-directional decoder model such that the hidden state at time t can in theory utilize all past/future information. The source/target word embedding matrices (\mathbf{X}, \mathbf{Y}) are shared between the inference network and the generative model.

An estimator for the gradient with respect to θ in this generative model is simple to obtain with Monte Carlo sampling (as usual). For the gradient with respect to ϕ ,

we experiment with two different estimators.

SCORE FUNCTION GRADIENT ESTIMATOR The first approach uses the score function gradient estimator from section 2.5.5 with a variance-reducing control variate from a soft attention model. Concretely, the inference network parameters have the following gradient expression,

$$\mathbb{E}_{q(z_t | x, y; \phi)} \left[\log p(y_t | x, y_{<t}, z_t; \theta) \times \nabla_{\phi} \log q(z_t | x, y; \phi) \right] - \nabla_{\phi} \text{KL}[q(z_t | x, y; \phi) \| p(z_t | x, y_{<t}; \theta)].$$

The second term in the above expression for the gradient can be calculated easily since it is simply the gradient of the KL divergence between two categorical distributions, which can be calculated exactly via enumeration,

$$\text{KL}[q(z_t | x, y; \phi) \| p(z_t | x, y_{<t}; \theta)] = \sum_{i=1}^S q(z_t = i | x, y; \phi) \log \frac{q(z_t = i | x, y; \phi)}{p(z_t = i | x, y_{<t}; \theta)}.$$

For the first term (i.e. the score function gradient estimator) we use a control variate B_t to reduce variance,

$$\mathbb{E}_{q(z_t | x, y; \phi)} \left[(\log p(y_t | x, y_{<t}; \theta) - B_t) \times \nabla_{\phi} \log q(z_t | x, y; \phi) \right],$$

where B_t is the output of the soft attention model at each time step,

$$\begin{aligned} B_t &= \log p_{\text{soft}}(y_t | x, y_{<t}; \theta_{\text{soft}}) \\ &= \log(\text{softmax}(\mathbf{W} \text{MLP}([\mathbf{h}_t; \mathbb{E}[\hat{\mathbf{s}}_t]]) + \mathbf{b})[y_t]) \end{aligned}$$

where θ_{soft} parameterizes a deterministic soft attention model. The soft attention model for producing the control variate (which has its own set of parameters) is trained alongside the hard attention model with each mini-batch. Intuitively, the variational posterior $q(z_t | x, y; \phi)$ is adjusted such that its samples give higher reward (i.e. likelihood) than a comparable soft attention model. In practice we estimate the above gradient expression with a single sample.

PATHWISE GRADIENT ESTIMATOR T The second approach uses a pathwise gradient estimator from a Gumbel-Softmax relaxation of the discrete distribution for the conditional likelihood, as described in section 4.2.3. First we reparameterize the conditional likelihood in terms of the one-hot vector representation of z_t . Letting \mathbf{z}_t be the one-hot representation of z_t , we then have

$$\begin{aligned} \log p(y_t | x, y, z_t; \theta) &= \log p(y_t | x, y, \mathbf{z}_t; \theta) \\ &= \log (\text{softmax}(\mathbf{W} \text{MLP}([\mathbf{h}_t; \mathbf{S} \mathbf{z}_t]) + \mathbf{b})) [y_t], \end{aligned}$$

where $\mathbf{S} \in \mathbb{R}^{d \times S}$ is the matrix obtained by stacking the source hidden states $[\mathbf{s}_1, \dots, \mathbf{s}_S]$.

Therefore the context vector is given by

$$\begin{aligned} \mathbf{S} \mathbf{z}_t &= \sum_{i=1}^S \mathbb{1}\{z_t = i\} \mathbf{s}_i \\ &= \bar{\mathbf{s}}_t. \end{aligned}$$

The variational distribution $q(\mathbf{z}_t | x, y; \phi)$ is also redefined in the obvious way. Letting $q_{\text{relax}}(\mathbf{u}_t | x, y; \phi, \tau)$ be the Gumbel-Softmax distribution over $\mathbf{u}_t \in \Delta^{S-1}$ with tempera-

ture τ , we are now ready to approximate the gradient as follows,

$$\begin{aligned}\nabla_{\phi} \mathbb{E}_{q(\mathbf{z}_t | x, y_{< t}; \phi)} [\log p(y_t | x, y_{< t}, \mathbf{z}_t; \theta)] &\approx \nabla_{\phi} \mathbb{E}_{q_{\text{relax}}(\mathbf{u}_t | x, y; \phi, \tau)} [\log p(y_t | x, y_{< t}, \mathbf{z}_t = \mathbf{u}_t; \theta)] \\ &= \mathbb{E}_{\mathbf{g} \sim \text{Gumbel}} [\nabla_{\phi} \log p(y_t | x, y_{< t}, \mathbf{z}_t = \text{softmax}((\log \boldsymbol{\alpha}_t + \mathbf{g})/\tau); \theta)],\end{aligned}$$

where $\boldsymbol{\alpha}_t = [\alpha_{t,1}, \dots, \alpha_{t,S}]$ is the vector of attention probabilities, $\mathbf{g} = [g_1, \dots, g_S]$ is the vector of i.i.d samples from $\text{Gumbel}(0, 1)$. The expectation can be approximated with a single sample to yield a low-variance (biased) estimator without the need for any control variates. The temperature term τ is annealed to a small value during training. Note that in deriving this estimator, it was crucial to work with the one-hot vector representation of \mathbf{z}_t instead of the original integer representation z_t in order to “relax” the conditional likelihood which was originally only defined for discrete variable z_t . This ensured that the following definition of log likelihood over y_t ,

$$\log p(y_t | x, y_{< t}, \mathbf{z}_t = \mathbf{u}_t) = \log (\text{softmax}(\mathbf{W} \text{MLP}([\mathbf{h}_t; \mathbf{S} \mathbf{u}_t]) + \mathbf{b})[y_t]),$$

was well-defined.

4.3.1 TEST TIME INFERENCE

The model is trained to maximize a lower bound on the log likelihood of target sentences $y^{(1:N)}$ conditioned on source sentences $x^{(1:N)}$. However, at test time we are only given the source sentence and must produce the most likely output under the model,

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

where \mathcal{Y} is the set of all sentences in the target language (up to some predefined maximum length).⁵ As the argmax is intractable to obtain exactly, a common strategy is to utilize beam search. For hard attention models we can marginalize over z_t at an additional multiplicative cost of $O(S)$ at each time step, which is feasible but expensive. We therefore propose a K -Max decoding strategy which approximates the prior $p(z_t | x, y_{<t}; \theta)$ with a sparser distribution that takes only the top K non-zero elements, i.e.,

$$p_{K\text{-Max}}(z_t = i | x, y_{<t}; \theta) \propto \mathbb{1}\{p(z_t = i | x, y_{<t}; \theta) \geq u_K\} \times p(z_t = i | x, y_{<t}; \theta),$$

where u_K is the K -th largest element of $p(z_t | x, y_{<t}; \theta)$. Then we estimate the marginal likelihood as,

$$p(y_t | x, y_{<t}; \theta) \approx \sum_{\{z_t: p_K(z_t | x, y_{<t}; \theta) > 0\}} p_{K\text{-Max}}(z_t | x, y_{<t}; \theta) \times p(y_t | x, y_{<t}; \theta),$$

and we use this approximated token-level distribution within standard beam search to obtain the approximate argmax for the hard attention model. This heuristic makes the multiplicative cost constant with respect to S , and we found this to work well empirically.

4.4 EMPIRICAL STUDY

4.4.1 EXPERIMENTAL SETUP

DATA Our experiments mainly utilize the IWSLT German to English dataset, which contains 153K/7K/7K train/validation/test set of parallel sentences of translated

⁵The inference network ϕ is only used for training and discarded at test time.

TED talks (Cettolo et al., 2014). This dataset is relatively small, but has become a standard benchmark for experimental neural machine translation models. We follow the same preprocessing as in Edunov et al. (2018) with a byte pair encoded vocabulary of 14k tokens (Sennrich et al., 2016) and train on sequences of length up to 125. To show that variational attention scales to large datasets, we also experiment on the WMT 2017 English to German dataset which has approximately 4M sentences (Bojar et al., 2017), utilizing the preprocessing in Vaswani et al. (2017). We use newstest2017 as the test set in this case. We evaluate using the BLEU score (Papineni et al., 2002), as is standard in machine translation.

HYPERPARAMETERS The encoder is a two-layer bi-directional LSTM with 512 units in each direction, and the decoder as a two-layer LSTM with 768 units. For the decoder, the convex combination of source hidden states at each time step from the attention distribution is used as additional input at the next time step (i.e. the input-feeding approach from Luong et al. (2015)). Word embeddings have 512 dimensions. The inference network consists of two bi-directional LSTMs (also two-layer and 512-dimensional) which is run over the source/target to obtain the hidden states at each time step. For the attention scoring function f , we combine the hidden states using bilinear attention (Luong et al., 2015) to produce the variational parameters. (In contrast the generative model uses MLP attention from (Bahdanau et al., 2015), though we saw little difference between the two parameterizations). Only the word embedding is shared between the inference network and the generative model.

Other training details include: batch size of 6, dropout rate of 0.3, parameter initialization over a uniform distribution $\mathcal{U}[-0.1, 0.1]$, gradient norm clipping at 5, and training for 30 epochs with Adam (learning rate = 0.0003, $\beta_1 = 0.9$, $\beta_2 = 0.999$) with

a learning rate decay schedule which starts halving the learning rate if validation perplexity does not improve. Most models converged well before 30 epochs. For decoding we use beam search with beam size 10 and length penalty $\alpha = 1$, from Wu et al. (2016). The length penalty added about 0.5 BLEU points across all the models.

BASELINES Experiments vary three components of the systems: (a) training objective (exact log marginal likelihood, lower bound from the prior, lower bound from the variational posterior), (b) training estimation (sampling vs. exact calculation of the expectation via enumeration, score function gradient estimator vs. pathwise gradient estimator with the relaxed Gumbel-Softmax distribution), and (c) test inference (exact enumeration vs K -max decoding). All models have the same architecture and the exact same number of parameters θ . For the score function gradient estimator trained with sampling, both the prior/variational models use the variance reducing control variate from soft attention.

CODE Our code is available at <https://github.com/harvardnlp/var-attn>.

4.4.2 RESULTS

Table 4.1 shows the main results. We first note that hard attention from the prior underperforms soft attention, even when the expectation in the gradient expression is calculated exactly via enumeration. This indicates that the previous negative results from hard attention can mostly be attributed to the large gap resulting from Jensen’s inequality, rather than approximation issues (such as high variance gradient estimators) from sampling. On the other hand, exact marginal likelihood outperforms soft attention, showing the benefits of explicitly modeling attention as a latent variable.

Model	Training Objective	∇ Estimation	PPL	BLEU
Soft Attn	$\log p(y x, \mathbb{E}_{p(z x)}[z])$	—	7.17	32.77
Marginal Likelihood	$\log \mathbb{E}_{p(z x)}[p(y x, z)]$	Enumeration	6.34	33.29
Hard Attn from Prior	$\mathbb{E}_{p(z x)}[p(y x, z)]$	Enumeration	7.37	31.40
Hard Attn from Prior	$\mathbb{E}_{p(z x)}[p(y x, z)]$	Score Function	7.38	31.00
Variational Attn	$\mathbb{E}_{q(z x,y)}[\log p(y x, z)] - \text{KL}$	Enumeration	6.08	33.68
Variational Attn	$\mathbb{E}_{q(z x,y)}[\log p(y x, z)] - \text{KL}$	Score Function	6.17	33.30
Variational Attn	$\mathbb{E}_{q_{\text{relax}}(\mathbf{u} x, y)}[\log p(y x, \mathbf{u})] - \text{KL}$	Pathwise	6.51	33.08

Table 4.1: Evaluation on IWSLT 2014 German-English test set for the various models. ∇ estimation indicates whether the gradients are calculated via enumeration or sampling. Sampled estimators can either be a score function estimator with control variate from a soft attention model, or a pathwise estimator from a Gumbel-Softmax relaxation to the discrete distribution. We evaluate intrinsically on perplexity (PPL) (lower is better) and extrinsically on BLEU (higher is better), where for BLEU we perform beam search with beam size 10 and length penalty with exact marginalization over z_t at each time step.

Surprisingly, variational attention with enumeration and score function gradients performs better than optimizing the exact log marginal likelihood, despite the fact that it is optimizing a lower bound. We believe that this is due to a posterior regularization effect on the generative model via the parameterization of the inference network, where the global BiLSTM parameterization over source and target for q provides beneficial inductive bias and aids generalization.^{6,7} We observe that training discrete attention with the pathwise gradient estimator from the relaxed, Gumbel-Softmax distribution is a viable strategy. This model also outperforms soft attention and slightly underperforms the model trained with enumeration/score function gradient estimator. Note that the pathwise gradient estimator does not require variance-reducing control variates from the soft attention model and is hence more memory

⁶Note that it is also possible to have $q(z)$ come from a pretrained external model, such as a traditional alignment model (Dyer et al., 2013).

⁷In chapter 5 we investigate this posterior regularization effect in greater detail where we use a more restrictive variational family to constrain the true posterior in order to encourage desired structures to emerge.

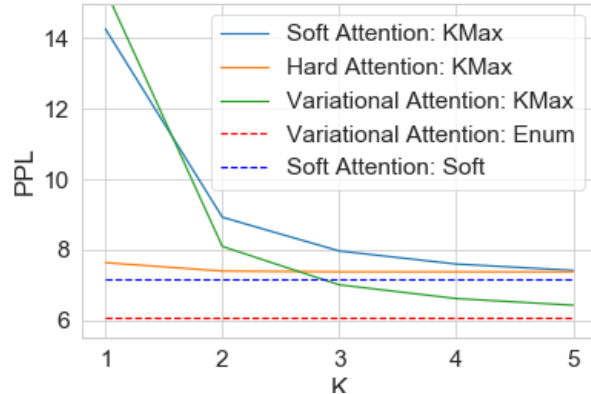


Figure 4.2: Test perplexity of different approaches while varying K to estimate $\log p(y_t | x, y_{<t}; \theta)$. Dotted lines compare soft baseline and variational with full enumeration.

efficient and faster to train than the score function gradient estimator.

On the larger WMT 2017 English-German task, our baseline soft attention reaches 24.10 BLEU score, while variational attention with the score function gradients reaches 24.98. This only reflects a reasonable setting without exhaustive hyperparameter tuning, yet we show that we variational attention can improve upon deterministic soft attention even in large data regimes.

4.4.3 ANALYSIS

For model analysis, we limit our scope and focus on the prior/variational attention models trained with the score function gradient estimator. Figure 4.2 shows the PPL of different models as we increase K . Good performance requires $K > 1$, but we only observe marginal benefits for $K > 5$. (We observed similar trends in BLEU). Thus the K -Max approximation during predictive inference is a viable strategy for efficient inference in hard attention models. We also observe that it is possible to *train* with soft attention and *test* using K -Max with a small performance drop (Soft KMax in Ta-

Model	$\mathbb{H}[p(z_t x, y_{<t}; \theta)]$
Soft Attention	1.24
Marginal Likelihood	0.82
Hard Attention from Prior	0.07
Variational Attention	0.52

Table 4.2: Entropy of the prior attention distributions $p(z_t | x, y_{<t}; \theta)$, averaged over target sequence length.

ble 4.2 (right)). This possibly indicates that soft attention models are approximating latent alignment models even though they are trained on a non-latent variable objective. On the other hand, we find that training with latent alignments and testing with soft attention performs poorly.

We next analyze the learned attention distributions in more detail. We show the average entropy of the prior attention distributions in Table 4.2, where we average across target sequence lengths. We observe that a model trained with hard attention from prior has very low entropy and is potentially overconfident. The opposite holds true for a model trained with soft attention, and the variational attention model falls somewhere in between. This is visualized in Figure 4.3 (right column), where we show the attention distribution between the soft/variational attention models over a fixed gold sentence. Besides performance, an advantage of these models is the ability to perform posterior inference, since the q function can be used directly to obtain posterior alignments. This is in contrast with hard attention using the prior, where the posterior is independent of the future information. Figure 4.3 shows the alignments of p and q for variational attention over a fixed sentence. We see that q is able to use future information to correct alignments. We note that the inability of soft and hard attention to produce good alignments has been noted as a major issue in NMT (Koehn & Knowles, 2017). While q is not used directly in left-to-right NMT de-

coding, it could be employed for other applications such as in an iterative refinement approach to sequence generation (Novak et al., 2016; Lee et al., 2018).

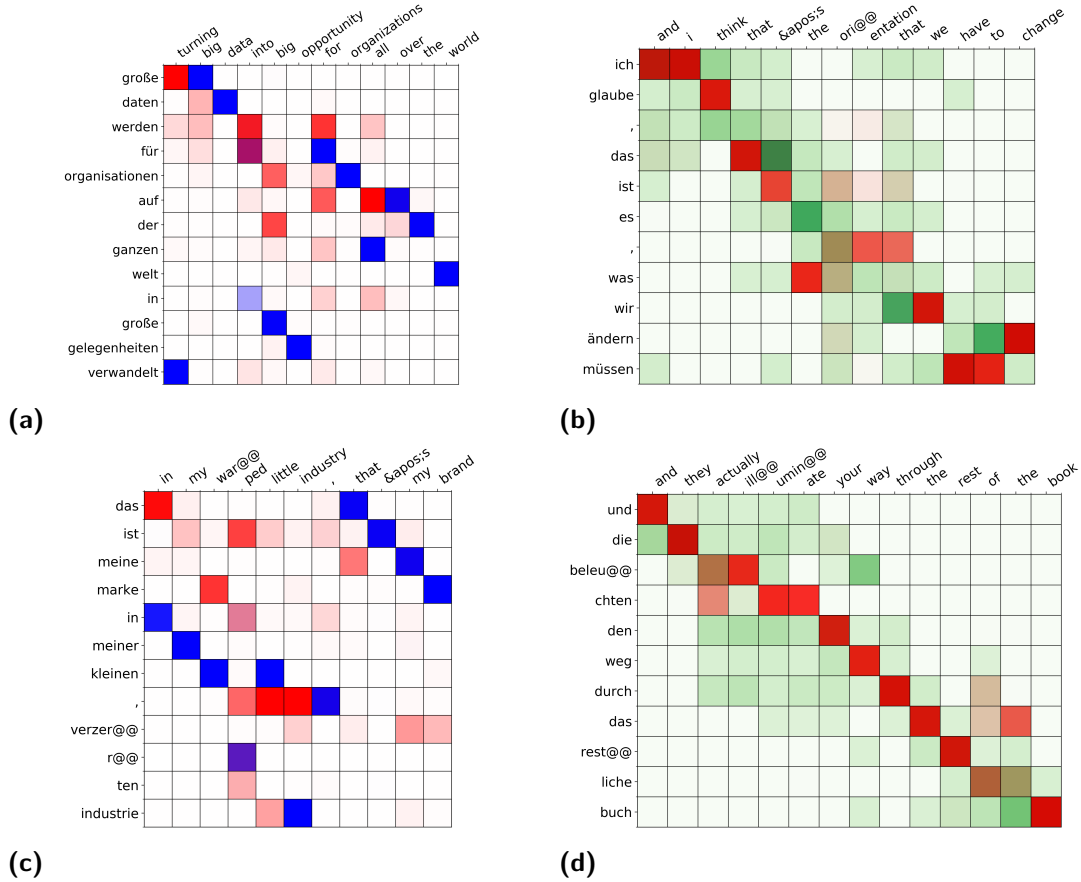


Figure 4.3: (Left Column) Examples highlighting the difference between the prior attention $p(z_t | x, y_{<t}; \theta)$ (red) and the variational posterior $q(z_t | x, y; \phi)$ (blue) when translating from German to English (left-to-right). The variational posterior is able to better handle reordering; in (a) the variational posterior successfully aligns ‘turning’ to ‘verwandelt’, in (c) we see a similar pattern with the alignment of the clause ‘that’s my brand’ to ‘das ist meine marke’. (Right Column) Comparisons between soft attention (green) and the prior attention from a model trained with variational attention (red). Alignments from both models are similar, but variational attention is lower entropy. Both soft and variational attention rely on aligning the inserted English word ‘orientation’ to the comma in (b) since a direct translation does not appear in the German source.

4.5 DISCUSSION

4.5.1 LIMITATIONS

Incorporating latent attention into neural models via the variational formulation is a promising alternative to deterministic soft attention. However, there are some practical limitations. Variational attention with the score function gradients needs a good control variate in the form of soft attention. We found this to be a necessary component for adequately training the system. This may prevent this technique from working when S is intractably large or when soft attention is not an option. Similarly, the Gumbel-Softmax approach with pathwise gradients requires that the generative model be well defined for a continuously relaxed discrete latent variable, which may not be applicable in general.

Contemporary architectures such as the Transformer (Vaswani et al., 2017) utilize many repeated attention models. For instance the current best translation models have the equivalent of 150 different attention queries per word translated. It is unclear if this approach can be used at that scale as predictive inference becomes combinatorial.

4.5.2 ATTENTION AS A LATENT VARIABLE

Soft attention was initially motivated as an approximation to latent alignment in machine translation,⁸ and is now an integral building block of contemporary deep architectures such as Transformer networks. However, its utility as a tool for model introspection is the subject of much current debate, as evidenced by a series of re-

⁸Indeed, the title of the original paper introducing soft attention is “Neural Machine Translation by Jointly Learning to Align and Translate” (Bahdanau et al., 2015).

cent papers that investigate soft attention’s ability (or lack thereof) to explain model outputs (Jain & Wallace, 2019; Serrano & Smith, 2019; Wiegrefe & Pinter, 2019). It would therefore be interesting to see if a latent variable formulation of attention can provide a more robust and probabilistically rigorous alternative to soft attention for model interpretability.

While our generative story interprets the categorical latent variable z_t as performing selection over the input (i.e., selecting which word to translate) at each time step, in practice we select over *contextualized* word representations $[\mathbf{s}_1, \dots, \mathbf{s}_S]$. Each vector \mathbf{s}_i can theoretically encode all information about the entire sentence x since it is the output from a bi-directional model. Thus, just because we have $z_t = i$ does not necessarily mean that the model is translating the particular word x_i . This is both a feature and a bug: it is a feature since it allows the model to softly select *phrases* with a single categorical latent variable, which are easier to work with in terms of inference (compared to, for example, binary vector-valued latent variables); it is a bug since it means that z_t cannot rigorously be interpreted as focusing on a single word (although in practice alignments do seem to correspond to human intuition at the word-level). An interesting future direction would involve working with latent variable versions of *structured* attention (Kim et al., 2017) which can explicitly attend to contiguous words (e.g., via a semi-Markov model), while at the same time employing encoders with Markov properties such that \mathbf{s}_i does not contain information about the entire source sentence.

4.6 RELATED WORK

Neural soft attention was originally introduced as an alternative approach for neural machine translation (Bahdanau et al., 2015), and have subsequently been successful on a wide range of tasks (see Cho et al. (2015) for a review of applications). Recent work has combined neural attention with traditional alignment (Cohn et al., 2016; Tu et al., 2016) and induced structure/sparsity (Martins & Astudillo, 2016; Kim et al., 2017; Liu & Lapata, 2017; Zhu et al., 2017; Niculae & Blondel, 2017; Niculae et al., 2018a; Mensch & Blondel, 2018), which can be combined with the variational approaches outlined in this paper.

In contrast to soft attention models, hard attention (Xu et al., 2015; Ba et al., 2015a) approaches use a single sample from a prior for training with a lower bound resulting from Jensen’s inequality. These models have proven much more difficult to train, and existing works typically treat hard attention as a black-box reinforcement learning problem with log likelihood as the reward (Xu et al., 2015; Ba et al., 2015a; Mnih et al., 2015; Gulcehre et al., 2016; Deng et al., 2017). Two notable exceptions are Ba et al. (2015b) and Lawson et al. (2018): both utilize amortized variational inference to learn a sampling distribution which is used to obtain importance-sampled estimates of the log marginal likelihood (Burda et al., 2015) via multiple samples. Our method uses different estimators and targets the single sample approach for efficiency.

There has also been significant work in using variational autoencoders for language modeling and machine translation. Of particular interest are those that augment an RNN with latent variables (typically Gaussian) at each time step (Chung et al., 2015; Fraccaro et al., 2016; Serban et al., 2017; Goyal et al., 2017a; Krishnan et al., 2017)

and those that incorporate latent variables into sequence-to-sequence models (Zhang et al., 2016; Bahuleyan et al., 2017; Su et al., 2018; Schulz et al., 2018). Our work differs by modeling an explicit model component (alignment) as a latent variable instead of auxiliary latent variables (e.g. topics). The term “variational attention” has also been used to refer to a different component of the output from attention (commonly called the context vector) as a latent variable (Bahuleyan et al., 2017), or to model both the memory and the alignment as a latent variable (Bornschein et al., 2017). Finally, contemporaneous work by Wu et al. (2018) and Shankar et al. (2018) also perform exact/approximate marginalization over latent alignments for sequence-to-sequence learning.

4.7 CONCLUSION

While attention networks are ubiquitous in natural language processing, they are difficult to use as latent variable models. In this chapter we explored alternative approaches to modeling alignment as a latent variable, and showed variational attention as a promising technique. In particular we saw that both the score function gradient estimator with a control variate derived from a soft attention model, and the path-wise gradient estimator derived from a Gumbel-Softmax relaxation of the discrete attention distribution, were viable methods for efficiently learning performant hard attention models. An important future direction will be to scale the method on more complex models such as multi-hop attention models, and to utilize these latent variables for interpretability and as a way to incorporate prior knowledge.

5

Latent Variable Model of Trees & Posterior Regularization

5.1 INTRODUCTION

This chapter considers a deep latent variable model of sentences where the latent variable corresponds to an unlabeled, binary parse tree of the sentence. Unlike the la-

The material in this chapter is adapted from [Kim et al. \(2019b\)](#).

tent variable models that we have seen in previous chapters, the latent variables in this chapter are *structured*, in that there is interdependence among the different latent components. In particular, we study unsupervised learning of recurrent neural network grammars (RNNG) (Dyer et al., 2016). RNNGs are generative models of language which jointly model syntax and surface structure by incrementally generating a syntax tree and sentence in a top-down, left-to-right order. Supervised RNNGs have been shown to outperform standard sequential language models, achieve excellent results on parsing (Dyer et al., 2016; Kuncoro et al., 2017), better encode syntactic properties of language (Kuncoro et al., 2018), and correlate with electrophysiological responses in the human brain (Hale et al., 2018). However, these all require annotated syntactic trees for training. In this chapter, we explore unsupervised learning of recurrent neural network grammars for language modeling and unsupervised parsing.

Past work on incorporating syntax into statistical models of language has focused on cases where either (i) the underlying structure is itself the goal (Klein & Manning, 2002), or (ii) the structure is useful in obtaining better language models (Emami & Jelinek, 2005). The goal of this chapter is to develop a model which obtains good language modeling performance and at the same time learns meaningful latent structure. We propose to achieve this by regularizing the posterior of a flexible generative model (which is crucial for good language modeling performance) through a form of posterior regularization (Ganchev et al., 2010) within a variational framework. In particular, we utilize a conditional random field (CRF) constituency parser (Finkel et al., 2008; Durrett & Klein, 2015) as an inference network, and find that it acts as a guide on the generative model through regularizing the posterior. We experiment with unsupervised RNNGs on English and Chinese and observe that they perform well as language models compared to their supervised counterparts and standard neural lan-

guage models. They also learn meaningful tree structures and outperform various baselines on unsupervised parsing.

5.2 BACKGROUND

We first give a high-level overview of recurrent neural network grammars as well as the problems they pose for unsupervised learning (5.2.1), deferring the exact parameterization of the RNNG to a later section (5.3.1). We then briefly review posterior regularization (5.2.2) and conditional random fields (5.2.3), both of which are crucial components of our approach.

5.2.1 RECURRENT NEURAL NETWORK GRAMMARS

Recurrent neural network grammars (RNNGs) model sentences by first generating a nested, hierarchical syntactic structure which is used to construct a context representation to be conditioned upon for upcoming words. Like neural language models, RNNGs make no independence assumptions. Instead they encode structural bias through operations that compose linguistic constituents through neural network-based composition functions. The lack of independence assumptions combined with syntactic constraints contribute to the strong language modeling performance. However they make unsupervised learning challenging.

First, marginalization is intractable. In our RNNG, the binary latent variable $z_t \in \{0, 1\}$ at each time step represents an action; whether to REDUCE and compose a constituent via merging the last two elements on the stack, or to generate the next word and SHIFT it onto the stack. Crucially, z_t fully depends on the previously gener-

ated actions $\mathbf{z}_{<t} = [z_1, \dots, z_{t-1}]$, which makes exact marginalization intractable.¹ Second, the biases imposed by the RNNG are relatively weak compared to those imposed by models like probabilistic context-free grammars (PCFGs). Even if exact marginalization were tractable, there is little pressure for non-trivial tree structure to emerge during unsupervised RNNG (URNNG) learning.

In this chapter we explore a technique for handling intractable marginalization while also injecting inductive bias for meaningful structure. Specifically we employ amortized variational inference (Kingma & Welling, 2014) with a structured inference network that *does* make significant independence assumptions, which encourages non-trivial structure through a form of posterior regularization.

5.2.2 POSTERIOR REGULARIZATION

Posterior regularization (Ganchev et al., 2010) describes a framework for injecting prior knowledge into probabilistic models via constraints on expectations of a model’s posterior. Letting $x \in \mathcal{X}$ be the observed data and $z \in \mathcal{Z}$ be the unobserved latent variable, one formulation of posterior regularization optimizes the following objective

$$\max_{\theta} \sum_{n=1}^N \log p(x^{(n)}; \theta) - \min_{q(z) \in \mathcal{Q}_{x^{(n)}}} \text{KL}[q(z) \parallel p(z \mid x^{(n)}; \theta)].$$

¹In contrast, in chapter 4 the attention at each time step (i.e. the categorical latent variable z_t) does not depend on any attention from previous time steps, which allowed us to directly calculate the log marginal likelihood via enumeration at a multiplicative (but not exponential) increase in cost.

Here \mathcal{Q}_x is the family of distributions over z (which depend on x) that satisfy the following

$$\mathcal{Q}_x = \{q(z) : \mathbb{E}_{q(z)}[\Phi(x, z)] \leq \mathbf{b}\},$$

where $\Phi : \mathcal{X} \times \mathcal{Z} \rightarrow \mathbb{R}^d$ is a feature function which encodes the desired constraints when combined with \mathbf{b} . Following the example in [Ganchev et al. \(2010\)](#), if

$$\Phi(x, z) = \text{“negative number of verbs in } z\text{”},$$

then

$$\mathcal{Q}_{x^{(n)}} = \{q(z) : \mathbb{E}_{q(z)}[\Phi(x^{(n)}, z)] \leq -1\}$$

expresses the constraint that each sentence should have at least one verb, in expectation. The generative model parameters θ are optimized against this posterior-regularized likelihood with EM-style coordinate ascent updates, which encourages the learned model to have posteriors that respect this constraint.² In many cases, prior knowledge can be more naturally expressed via constraints on data-dependent posteriors, and thus posterior regularization provides an attractive alternative to Bayesian priors for injecting prior knowledge into latent variable models.

²We give a greatly simplified exposition here for brevity. See the original formulation in [Ganchev et al. \(2010\)](#) for full details and extensions, e.g. adding slack variables to allow for constraint violations.

5.2.3 CONDITIONAL RANDOM FIELDS

A conditional random field (CRF) (Lafferty et al., 2001) specifies a distribution over discrete structure z given observed variable x via a globally normalized Gibbs distribution,

$$p(z | x; \theta) = \frac{1}{Z(x; \theta)} \prod_{z_c \in \mathcal{C}(x, z)} \exp(\psi_c(x, z_c; \theta)),$$

where $\mathcal{C}(x, z)$ is the set of cliques in a graphical model over x and z , $\psi_c(x, z_c; \theta)$ is the score (i.e. log potential) for the substructure (x, z_c) , and

$$Z(x; \theta) = \sum_{z' \in \mathcal{Z}} \prod_{z_c \in \mathcal{C}(x, z')} \exp(\psi_c(x, z_c; \theta)),$$

is the partition function which guarantees that $p(z | x; \theta)$ is a valid probability distribution over \mathcal{Z} .

The score function $\psi_c(x, z_c; \theta)$ can itself be parameterized as a neural network. CRFs and their neural extensions have been widely utilized for structured prediction tasks in NLP, from part-of-speech tagging to named entity recognition to parsing (Lafferty et al., 2001; Finkel et al., 2008; Collobert et al., 2011; Durrett & Klein, 2015; Lample et al., 2016; Ma & Hovy, 2016). For our purposes we are interested in a CRF that gives rise to a distribution over unlabeled binary trees,

$$p(z | x; \theta) \propto \mathbb{1}\{z \text{ is a valid binary tree}\} \prod_{i \leq j} \exp(\psi(x, z_i, z_j; \theta)),$$

where $\psi(x, z_i, z_j; \theta)$ is a score for span $[x_i, x_{i+1}, \dots, x_j]$'s being a constituent. Under this model we can calculate the partition function in $O(T^3)$ time (where T is the sen-

tence length) with the standard inside algorithm (Baker, 1979).

5.3 UNSUPERVISED RECURRENT NEURAL NETWORK GRAMMARS

We now describe our generative model (an RNNG) and the structured inference network in more detail. We use $x = [x_1, \dots, x_T]$ to denote a sentence of length T , and $\mathbf{z} \in \mathcal{Z}_T$ to denote an unlabeled binary parse tree over a sequence of length T , represented as a binary vector $[z_1, \dots, z_{2T-1}]$ of length $2T - 1$.³ Here 0 and 1 correspond to SHIFT and REDUCE actions, explained below.

5.3.1 GENERATIVE MODEL

An RNNG defines a joint probability distribution $p(x, \mathbf{z}; \theta)$ over sentences x and parse trees \mathbf{z} . We consider a simplified version of the original RNNG (Dyer et al., 2016) by ignoring constituent labels and only considering binary trees. The RNNG utilizes an RNN to parameterize a stack data structure (Dyer et al., 2015) of partially-completed constituents to incrementally build the parse tree while generating terminals. Using the current stack representation, the model samples an action (SHIFT or REDUCE) at each time step: SHIFT generates a terminal symbol (word) and shifts it onto the stack,⁴ REDUCE pops the last two elements off the stack, composes them, and shifts the composed representation onto the stack.

Formally, let $S = [(\mathbf{0}, \mathbf{0})]$ be the initial stack. Each item of the stack will be a

³The cardinality of $\mathcal{Z}_T \subset \{0, 1\}^{2T-1}$ is given by the $(T - 1)$ -th Catalan number, i.e.

$$|\mathcal{Z}_T| = \frac{(2T - 2)!}{T!(T - 1)!}.$$

⁴A better name for SHIFT would be GENERATE (as in Dyer et al. (2016)), but we use SHIFT to emphasize similarity with the shift-reduce parsing.

pair, where the first element is the hidden state of the stack LSTM, and the second element is an input vector, described below. We use $\text{top}(S)$ to refer to the top pair in the stack. The push and pop operations are defined imperatively in the usual way. At each time step, the next action z_t (SHIFT or REDUCE) is sampled from a Bernoulli distribution parameterized in terms of the current stack representation. Letting $(\mathbf{h}_{\text{prev}}, \mathbf{g}_{\text{prev}}) = \text{top}(S)$, we have

$$z_t \sim \text{Bernoulli}(p_t),$$

$$p_t = \sigma(\mathbf{w}^\top \mathbf{h}_{\text{prev}} + b).$$

Subsequent generation depend on z_t :

- If $z_t = 0$ (SHIFT), the model first generates a terminal symbol via sampling from a categorical distribution whose parameters come from an affine transformation and a softmax,

$$y \sim \text{softmax}(\mathbf{W}\mathbf{h}_{\text{prev}} + \mathbf{b}).$$

Then the generated terminal is shifted onto the stack using a stack LSTM,

$$\mathbf{h}_{\text{next}} = \text{LSTM}(\mathbf{e}_y, \mathbf{h}_{\text{prev}}),$$

$$\text{push}(S, (\mathbf{h}_{\text{next}}, \mathbf{e}_y)),$$

where \mathbf{e}_y is the word embedding for the word y .

- If $z_t = 1$ (REDUCE), we pop the last two elements off the stack,

$$(\mathbf{h}_r, \mathbf{g}_r) = \text{pop}(S), \quad (\mathbf{h}_l, \mathbf{g}_l) = \text{pop}(S),$$

and obtain a new representation that combines the left/right constituent representations using a tree LSTM (Tai et al., 2015; Zhu et al., 2015),

$$\mathbf{g}_{\text{new}} = \text{TreeLSTM}(\mathbf{g}_l, \mathbf{g}_r).$$

Note that we use \mathbf{g}_l and \mathbf{g}_r to obtain the new representation instead of \mathbf{h}_l and \mathbf{h}_r .⁵ We then update the stack using \mathbf{g}_{new} ,

$$\begin{aligned} (\mathbf{h}_{\text{prev}}, \mathbf{g}_{\text{prev}}) &= \text{top}(S), \\ \mathbf{h}_{\text{new}} &= \text{LSTM}(\mathbf{g}_{\text{new}}, \mathbf{h}_{\text{prev}}), \\ \text{push}(S, (\mathbf{h}_{\text{new}}, \mathbf{g}_{\text{new}})). \end{aligned}$$

The generation process continues until an end-of-sentence symbol is generated. For a sentence $x = [x_1, \dots, x_T]$ of length T , the binary parse tree is given by the binary vector $\mathbf{z} = [z_1, \dots, z_{2T-1}]$.⁶

In this model, the joint log likelihood decomposes as a sum of terminal/action log likelihoods,

$$\log p(x, \mathbf{z}; \theta) = \underbrace{\sum_{j=1}^T \log p(x_j | x_{<j}, \mathbf{z}_{<n(j)}; \theta)}_{\log p(x | \mathbf{z}; \theta)} + \underbrace{\sum_{t=1}^{2T-1} \log p(z_t | x_{<m(t)}, \mathbf{z}_{<t}; \theta)}_{\log p(\mathbf{z} | x_{<\mathbf{z}}; \theta)},$$

⁵The update equations for the tree LSTM (and the stack LSTM) also involve *cell* states in addition to the hidden states. To reduce notational clutter we do not explicitly show the cell states and instead subsume them into \mathbf{g} . If one (or both) of the inputs to the tree LSTM is a word embedding, the associated cell state is taken to be zero. See Tai et al. (2015) for the exact parameterization.

⁶As it stands, the support of \mathbf{z} is $\{0, 1\}^{2T-1}$, all binary vectors of length $2T - 1$. To restrict our distribution to \mathcal{Z}_T (binary vectors which describe valid trees), we constrain z_t to be valid at each time step, which amounts to deterministically choosing $z_t = 0$ (SHIFT) if there are fewer than two elements (not counting the initial zero tuple) on the stack.

where $\mathbf{z}_{<n(j)}$ refers to all actions before generating the j -th word, and similarly $\mathbf{x}_{<m(t)}$ refers to all words generated before taking the t -th action. For brevity, from here on we will use $\log p(x | \mathbf{z}; \theta)$ to refer to the first term (terminal log likelihood) and $\log p(\mathbf{z} | x_{<\mathbf{z}}; \theta)$ to refer to the second term (action log likelihood) in the above decomposition.

In the supervised case where ground-truth \mathbf{z} is available, we can straightforwardly perform gradient-based optimization to maximize the joint log likelihood $\log p(x, \mathbf{z}; \theta)$. In the unsupervised case, the standard approach is to maximize the log marginal likelihood,

$$\log p(x; \theta) = \log \sum_{\mathbf{z}' \in \mathcal{Z}_T} p(x, \mathbf{z}'; \theta).$$

However this summation is intractable because z_t fully depends on all previous actions $[z_1, \dots, z_{t-1}]$. Even if this summation were tractable, it is not clear that meaningful latent structures would emerge given the lack of explicit independence assumptions in the RNNG (e.g. it is clearly not context-free). Indeed, in early experiments we tried training the model via marginalizing over the trees via enumeration on short sentences of up to length 10, but did not observe any meaningful structures to emerge.⁷ We handle these issues with amortized variational inference with a form of posterior regularization from a structured inference network parameterized as a CRF parser.

5.3.2 POSTERIOR REGULARIZATION WITH CONDITIONAL RANDOM FIELDS

Consider an inference network ϕ that parameterizes $q(\mathbf{z} | x; \phi)$, a variational posterior distribution over parse trees \mathbf{z} given the sentence x . As in previous chapters, this distribution is used to form an evidence lower bound (ELBO) on the log marginal

⁷We found the model to collapse to fully left or right branching trees depending on hyperparameters.

likelihood,

$$\text{ELBO}(\theta, \phi; x) = \mathbb{E}_{q(\mathbf{z} | x; \phi)} \left[\log \frac{p(x, \mathbf{z}; \theta)}{q(\mathbf{z} | x; \phi)} \right].$$

We maximize the ELBO with respect to both model parameters θ and inference network parameters ϕ . We saw in 2.5.3 that rearranging the ELBO gives the following optimization problem for the entire dataset,

$$\max_{\theta, \phi} \sum_{n=1}^N \log p(x^{(n)}; \theta) - \text{KL}[q(\mathbf{z} | x^{(n)}; \phi) \| p(\mathbf{z} | x^{(n)}; \theta)].$$

From this formulation of the ELBO, we notice its similarity to the posterior regularization objective from 5.2.2. In particular, ϕ is trained to match the variational posterior $q(\mathbf{z} | x; \phi)$ to the true posterior $p(\mathbf{z} | x; \theta)$, but θ is *also* trained to match the true posterior to the variational posterior. Indeed, there is some evidence to suggest that generative models trained with amortized variational inference (i.e. variational autoencoders) learn posterior distributions that are close to the variational family (Burda et al., 2015; Cremer et al., 2018).

We can use this to our advantage with an inference network that injects inductive bias through conditional independence properties (conditioned on x). We propose to do this by using a context-free model for the inference network, in particular, a neural CRF parser (Durrett & Klein, 2015; Liu et al., 2018). This choice can be seen as a form of posterior regularization that limits posterior flexibility of the overly powerful RNNG generative model. Concretely, observe that if we optimize the posterior regu-

larization objective from 5.2.2

$$\max_{\theta} \sum_{n=1}^N \log p(x^{(n)}; \theta) - \min_{q(\mathbf{z}) \in \mathcal{Q}_{x^{(n)}}} \text{KL}[q(\mathbf{z}) \parallel p(\mathbf{z} | x^{(n)}; \theta)],$$

$$\mathcal{Q}_x = \{q(\mathbf{z}) : \mathbb{E}_{q(\mathbf{z})}[\Phi(x, \mathbf{z})] \leq \mathbf{b}\},$$

with coordinate ascent style updates (i.e. update q , then update θ), then it is equivalent to variational EM where we take the variational family to be the distributions over \mathbf{z} that respect the linearity constraints $\mathbb{E}_{q(\mathbf{z})}[\Phi(x, \mathbf{z})] \leq \mathbf{b}$. Conversely, the ELBO objective with a CRF inference network is then roughly equivalent to the posterior regularization objective where we take \mathcal{Q}_x to be distributions over \mathbf{z} such that (1) context-free assumptions hold conditioned on x and (2) log potentials (i.e. the span scores) are from a global neural network. Note, however, that the conditional independence assumptions in the CRF cannot be characterized through linear constraints over posterior expectations. Hence our use of the term “posterior regularization” is in a slightly different sense than in the original work of [Ganchev et al. \(2010\)](#). In preliminary experiments, we also attempted to learn latent trees with a transition-based parser (which does not make explicit conditional independence assumptions) that conditions on the entire sentence. However we found that under this setup, the inference network degenerated into a local minimum whereby it always generated trivial trees despite various optimization strategies.⁸

⁸[Williams et al. \(2018\)](#) observe a similar phenomenon in the context of learning latent trees for classification tasks. However [Li et al. \(2019a\)](#) find that it is possible use a transition-based parser as the inference network for dependency grammar induction, if the inference network is constrained via posterior regularization based on universal syntactic rules ([Naseem et al., 2010](#)).

Algorithm 1 Inside algorithm for calculating $Z_T(x)$

```
1: procedure INSIDE( $s$ )                                ▷ scores  $s_{ij}$  for  $i \leq j$ 
2:   for  $i := 1$  to  $T$  do                                ▷ length-1 spans
3:      $\beta[i, i] = \exp(s_{ii})$ 
4:   end for
5:   for  $\ell := 1$  to  $T - 1$  do                                ▷ span length
6:     for  $i := 1$  to  $T - \ell$  do                                ▷ span start
7:        $j = i + \ell$                                 ▷ span end
8:        $\beta[i, j] = \sum_{k=i}^{j-1} \exp(s_{ij}) \cdot \beta[i, k] \cdot \beta[k + 1, j]$ 
9:     end for
10:  end for
11:  return  $\beta[1, T]$                                 ▷ return partition function  $Z_T(x)$ 
12: end procedure
```

NEURAL CRF PARAMETERIZATION The parameterization of span scores in the inference network is similar to recent works (Wang & Chang, 2016; Stern et al., 2017; Kitaev & Klein, 2018): we add position embeddings to word embeddings and run a bidirectional LSTM over the input representations to obtain the forward $[\vec{\mathbf{h}}_1, \dots, \vec{\mathbf{h}}_T]$ and backward $[\overleftarrow{\mathbf{h}}_1, \dots, \overleftarrow{\mathbf{h}}_T]$ hidden states. The score $s_{ij} \in \mathbb{R}$ for a constituent spanning x_i to x_j is given by,

$$s_{ij} = \text{MLP}([\vec{\mathbf{h}}_{j+1} - \vec{\mathbf{h}}_i; \overleftarrow{\mathbf{h}}_{i-1} - \overleftarrow{\mathbf{h}}_j]).$$

Then, letting \mathbf{B} be the binary matrix representation of a tree ($\mathbf{B}_{ij} = 1$ means there is a constituent spanning x_i and x_j), the CRF parser defines a distribution over binary trees via the Gibbs distribution,

$$q(\mathbf{B} | x; \phi) = \frac{1}{Z_T(x)} \exp \left(\sum_{i \leq j} \mathbf{B}_{ij} s_{ij} \right),$$

where $Z_T(x)$ is the partition function,

$$Z_T(x) = \sum_{\mathbf{B}' \in \mathcal{B}_T} \exp \left(\sum_{i \leq j} \mathbf{B}'_{ij} s_{ij} \right),$$

and ϕ denotes the parameters of the inference network (i.e. the bidirectional LSTM and the MLP). Calculating $Z_T(x)$ requires a summation over an exponentially-sized set $\mathcal{B}_T \subset \{0, 1\}^{T \times T}$, the set of all binary matrices that represent valid binary trees over a length T sequence. We can perform the summation in $O(T^3)$ using the inside algorithm (Baker, 1979), shown in Algorithm 1. This computation is itself differentiable and amenable to gradient-based optimization. Finally, letting $f : \mathcal{B}_T \rightarrow \mathcal{Z}_T$ be the bijection between the binary tree matrix representation and a sequence of SHIFT/REDUCE actions, the CRF inference network defines a distribution over \mathcal{Z}_T via $q(\mathbf{z} | x; \phi) \triangleq q(f^{-1}(\mathbf{z}) | x; \phi)$. Figure 5.1 gives an overview of our approach.

5.3.3 LEARNING AND INFERENCE

For training, we use the following variant of the ELBO,

$$\text{ELBO}(\theta, \phi; x) = \mathbb{E}_{q(\mathbf{z} | x; \phi)} [\log p(x, \mathbf{z}; \theta)] + \mathbb{H}[q(\mathbf{z} | x; \phi)],$$

where

$$\mathbb{H}[q(\mathbf{z} | x; \phi)] = \mathbb{E}_{q(\mathbf{z} | x; \phi)} [-\log q(\mathbf{z} | x; \phi)],$$

is the entropy of the variational posterior. A Monte Carlo estimate for the gradient

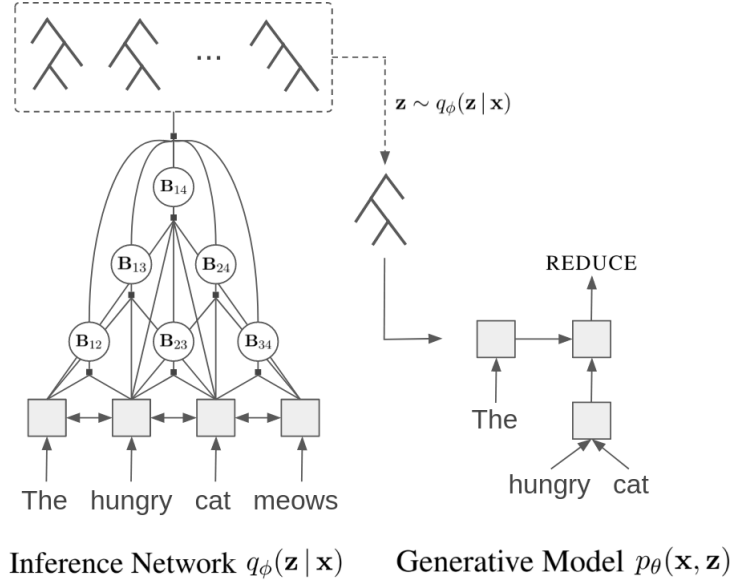


Figure 5.1: Overview of our approach. The inference network $q(\mathbf{z} | x; \phi)$ (left) is a CRF parser which produces a distribution over binary trees (shown in dotted box). \mathbf{B}_{ij} are random variables for existence of a constituent spanning i -th and j -th words, whose potentials are the output from a bidirectional LSTM (the global factor ensures that the distribution is only over valid binary trees). The generative model $p(x, \mathbf{z}; \theta)$ (right) is an RNNG which consists of a stack LSTM (from which actions/words are predicted) and a tree LSTM (to obtain constituent representations upon REDUCE). Training involves sampling a binary tree from $q(\mathbf{z} | x; \phi)$, converting it to a sequence of shift/reduce actions ($\mathbf{z} = [\text{SHIFT}, \text{SHIFT}, \text{SHIFT}, \text{REDUCE}, \text{REDUCE}, \text{SHIFT}, \text{REDUCE}]$ in the above example), and optimizing the log joint likelihood $\log p(x, \mathbf{z}; \theta)$.

with respect to θ is

$$\nabla_{\theta} \text{ELBO}(\theta, \phi; x) \approx \frac{1}{K} \sum_{k=1}^K \nabla_{\theta} \log p(x, \mathbf{z}^{(k)}; \theta),$$

with samples $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(K)}$ from $q(\mathbf{z} | x; \phi)$. Sampling uses the intermediate values calculated during the inside algorithm to sample split points recursively (Goodman, 1998; Finkel et al., 2006), as shown in Algorithm 2. The gradient with respect to ϕ involves two parts. The entropy term $\mathbb{H}[q(\mathbf{z} | x; \phi)]$ can be calculated exactly in $O(T^3)$, again using the intermediate values from the inside algorithm (see Algorithm 3),

Algorithm 2 Top-down sampling a tree from $q(\mathbf{z} \mid x; \phi)$

```
1: procedure SAMPLE( $\beta$ ) ▷  $\beta$  from running INSIDE(s)
2:    $\mathbf{B} = \mathbf{0}$  ▷ binary matrix representation of tree
3:    $Q = [(1, T)]$  ▷ queue of constituents
4:   while  $Q$  is not empty do
5:      $(i, j) = \text{pop}(Q)$ 
6:      $\tau = \sum_{k=i}^{j-1} \beta[i, k] \cdot \beta[k+1, j]$ 
7:     for  $k := i$  to  $j-1$  do ▷ get distribution over splits
8:        $w_k = (\beta[i, k] \cdot \beta[k+1, j]) / \tau$ 
9:     end for
10:     $k \sim \text{Cat}([w_i, \dots, w_{j-1}])$  ▷ sample a split point
11:     $\mathbf{B}_{i,k} = 1, \mathbf{B}_{k+1,j} = 1$  ▷ update  $\mathbf{B}$ 
12:    if  $k > i$  then ▷ if left child has width  $> 1$ 
13:      push( $Q, (i, k)$ ) ▷ add to queue
14:    end if
15:    if  $k+1 < j$  then ▷ if right child has width  $> 1$ 
16:      push( $Q, (k+1, j)$ ) ▷ add to queue
17:    end if
18:  end while
19:   $\mathbf{z} = f(\mathbf{B})$  ▷  $f : \mathcal{B}_T \rightarrow \mathcal{Z}_T$  maps matrix representation of tree to sequence of actions.
20:  return  $\mathbf{z}$ 
21: end procedure
```

where we adapt the algorithm for calculating tree entropy in PCFGs from Hwa (2000) to the CRF case. Since each step of this dynamic program is differentiable, we can obtain the gradient $\nabla_{\phi} \mathbb{H}[q(\mathbf{z} \mid x; \phi)]$ using automatic differentiation.⁹ An estimator for the gradient with respect to $\mathbb{E}_{q(\mathbf{z} \mid x; \phi)}[\log p(x, \mathbf{z}; \theta)]$ is obtained via the score function

⁹Note that $\nabla_{\phi} \mathbb{H}[q(\mathbf{z} \mid x; \phi)]$ can also be computed using the inside-outside algorithm and a second-order expectation semiring (Li & Eisner, 2009), which has the same asymptotic runtime complexity but generally better constants.

gradient estimator (Glynn, 1987; Williams, 1992),

$$\begin{aligned}\nabla_{\phi} \mathbb{E}_{q(\mathbf{z} | x; \phi)} [\log p(x, \mathbf{z}; \theta)] &= \mathbb{E}_{q(\mathbf{z} | x; \phi)} [\log p(x, \mathbf{z}; \theta) \nabla_{\phi} \log q(\mathbf{z} | x; \phi)] \\ &\approx \frac{1}{K} \sum_{k=1}^K \log p(x, \mathbf{z}^{(k)}; \theta) \nabla_{\phi} \log q(\mathbf{z}^{(k)} | x; \phi).\end{aligned}$$

As mentioned in 2.5.5, the above estimator is unbiased but typically suffers from high variance. To reduce variance, we use a control variate derived from an average of the other samples’ joint likelihoods (Mnih & Rezende, 2016), yielding the following estimator,

$$\frac{1}{K} \sum_{k=1}^K (\log p(x, \mathbf{z}^{(k)}; \theta) - r^{(k)}) \nabla_{\phi} \log q(\mathbf{z}^{(k)} | x; \phi),$$

where

$$r^{(k)} = \frac{1}{K-1} \sum_{j \neq k} \log p(x, \mathbf{z}^{(j)}; \theta).$$

This control variate worked better than control variates from an auxiliary network (Mnih & Gregor, 2014; Deng et al., 2018) or a language model (Yin et al., 2018).

5.4 EMPIRICAL STUDY

5.4.1 EXPERIMENTAL SETUP

DATA For English we use the Penn Treebank (Marcus et al., 1993, PTB) with splits and preprocessing from Dyer et al. (2016) which retains punctuation and replaces singleton words with Berkeley parser’s mapping rules, resulting in a vocabulary of 23,815 word types. Notably this vocabulary size is much larger than the standard PTB LM setup from Mikolov et al. (2010) which uses 10K types. Also different from the LM setup, we model each sentence separately instead of carrying information across sen-

Algorithm 3 Calculating the tree entropy $\mathbb{H}[q(\mathbf{z} \mid x; \phi)]$

```
1: procedure ENTROPY( $\beta$ ) ▷  $\beta$  from running INSIDE(s)
2:   for  $i := 1$  to  $T$  do ▷ initialize entropy table
3:      $H[i, i] = 0$ 
4:   end for
5:   for  $l := 1$  to  $T - 1$  do ▷ span length
6:     for  $i := 1$  to  $T - l$  do ▷ span start
7:        $j = i + l$  ▷ span end
8:        $\tau = \sum_{u=i}^{j-1} \beta[i, u] \cdot \beta[u + 1, j]$ 
9:       for  $u := i$  to  $j - 1$  do
10:         $w_u = (\beta[i, u] \cdot \beta[u + 1, j]) / \tau$ 
11:      end for
12:       $H[i, j] = \sum_{u=i}^{j-1} (H[i, u] + H[u + 1, j] - \log w_u) \cdot w_u$ 
13:    end for
14:  end for
15:  return  $H[1, T]$  ▷ return tree entropy  $\mathbb{H}[q(\mathbf{z} \mid x; \phi)]$ 
16: end procedure
```

tence boundaries, as the RNNG is a generative model of sentences. Hence our perplexity numbers are not comparable to the standard PTB LM results (Melis et al., 2018b; Merity et al., 2018; Yang et al., 2018).

Since the PTB is rather small, and since the URNNG does not require annotation, we also test our approach on a subset of the one billion word corpus (Chelba et al., 2013). We randomly sample 1M sentences for training and 2K sentences for validation/test, and limit the vocabulary to 30K word types. While still a subset of the full corpus (which has 30M sentences), this dataset is two orders of magnitude larger than PTB. Experiments on Chinese utilize version 5.1 of the Chinese Penn Treebank (CTB) (Xue et al., 2005), with the same splits as in Chen & Manning (2014). Singleton words are replaced with a single $\langle \text{UNK} \rangle$ token, resulting in a vocabulary of 17,489 word types.

HYPERPARAMETERS The stack LSTM has two layers with input/hidden size equal to 650 and dropout of 0.5. The tree LSTM also has 650 units. The inference network uses a one-layer bidirectional LSTM with 256 hidden units, and the MLP (to produce span scores s_{ij} for $i \leq j$) has a single hidden layer with a ReLU nonlinearity followed by layer normalization (Ba et al., 2016) and dropout of 0.5. We share word embeddings between the generative model and the inference network, and also tie weights between the input/output word embeddings (Press & Wolf, 2016).

Optimization of the model itself required standard techniques for avoiding posterior collapse in VAEs, where posterior collapse in our context means that $q(\mathbf{z} | x; \phi)$ always produced trivial (always left or right branching) trees. We warm-up the ELBO objective by linearly annealing (per batch) the weight on the conditional prior $\log p(\mathbf{z} | x_{<\mathbf{z}}; \theta)$ and the entropy $\mathbb{H}[q(\mathbf{z} | x; \phi)]$ from 0 to 1 over the first two epochs. This is analogous to KL-annealing in VAEs with continuous latent variables in chapter 3 (Bowman et al., 2016; Sønderby et al., 2016). We train for 18 epochs (enough for convergence for all models) with a batch size of 16 and $K = 8$ samples for the Monte Carlo gradient estimators. The generative model is optimized with SGD with learning rate equal to 1, except for the affine layer that produces a distribution over the actions, which has learning rate 0.1. Gradients of the generative model are clipped at 5. The inference network is optimized with Adam with learning rate 0.0001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and gradient clipping at 1. As Adam converges significantly faster than SGD (even with a much lower learning rate), we stop training the inference network after the first two epochs. We found it important to utilize separate optimizers for the inference network/generative model.

Initial model parameters are sampled uniformly from $\mathcal{U}(-0.1, 0.1)$. The learning rate starts decaying by a factor of 2 each epoch after the first epoch at which valida-

tion performance does not improve, but this learning rate decay is not triggered for the first eight epochs to ensure adequate training. We use the same hyperparameters/training setup for both PTB and CTB. For experiments on (the subset of) the one billion word corpus, we use a smaller dropout rate of 0.1. The corresponding baseline language model also uses the smaller dropout rate.

All models are trained with an end-of-sentence token, but for perplexity calculation these tokens are not counted to be comparable to prior work (Dyer et al., 2016; Kunzoro et al., 2017; Buys & Blunsom, 2018). To be more precise, the inference network does not make use of the end-of-sentence token to produce parse trees, but the generative model is trained to generate the end-of-sentence token after the final REDUCE operation.

BASELINES We compare the unsupervised RNNG (URNNG) against several baselines: (1) RNNLM, a standard LSTM language model whose size is the same as URNNG’s stack LSTM; (2) Parsing Reading Predict Network (PRPN) (Shen et al., 2018b), a neural language model that uses gated attention layers to embed soft tree-like structures into a neural network; (3) RNNG with trivial trees (left branching, right branching, random); (4) supervised RNNG trained on unlabeled, binarized gold trees.¹⁰ Note that the supervised RNNG also trains a discriminative parser $q(\mathbf{z} | x; \phi)$ (alongside the generative model $p(x, \mathbf{z}; \theta)$) in order to sample parse forests for perplexity evaluation (i.e. importance sampling). This discriminative parser has the same architecture as URNNG’s inference network. For all models, we perform early stopping

¹⁰We use right branching binarization—Matsuzaki et al. (2005) find that differences between various binarization schemes have marginal impact. Our supervised RNNG therefore differs from the original RNNG, which trains on non-binarized trees and does not ignore constituent labels.

Model	PTB		CTB	
	PPL	F_1	PPL	F_1
RNNLM	93.2	–	201.3	–
PRPN (default)	126.2	32.9	290.9	32.9
PRPN (tuned)	96.7	41.2	216.0	36.1
Left Branching Trees	100.9	10.3	223.6	12.4
Right Branching Trees	93.3	34.8	203.5	20.6
Random Trees	113.2	17.0	209.1	17.4
URNNG	90.6	40.7	195.7	29.1
RNNG	88.7	68.1	193.1	52.3
RNNG \rightarrow URNNG	85.9	67.7	181.1	51.9
Oracle Binary Trees	–	82.5	–	88.6

Table 5.1: Language modeling perplexity (PPL) and grammar induction F_1 scores on English (PTB) and Chinese (CTB) for the different models. We separate results for those that make do not make use of annotated data (top) versus those that do (mid). Note that our PTB setup from [Dyer et al. \(2016\)](#) differs considerably from the usual language modeling setup ([Mikolov et al., 2010](#)) since we model each sentence independently and use a much larger vocabulary.

based on validation perplexity.

CODE Our code is available at <https://github.com/harvardnlp/urnng>.

5.4.2 RESULTS

LANGUAGE MODELING Table 5.1 shows perplexity for the different models on PTB/CTB.

As a language model URNNG outperforms an RNNLM and is competitive with the supervised RNNG, where for RNNG and URNNG we estimate the log marginal likelihood (and hence, perplexity) with $K = 1000$ importance-weighted samples,

$$\log p(x; \theta) \approx \log \left(\frac{1}{K} \sum_{k=1}^K \frac{\log p(x, \mathbf{z}^{(k)}; \theta)}{q(\mathbf{z}^{(k)} | x; \phi)} \right).$$

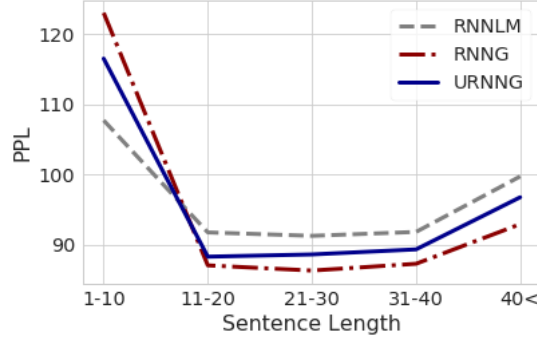


Figure 5.2: Perplexity of the different models grouped by sentence length on PTB.

During evaluation only, we also flatten $q(\mathbf{z} | x; \phi)$ by dividing span scores s_{ij} by a temperature term 2.0 before feeding it to the CRF. The left branching baseline performs poorly, implying that the strong performance of URNNG/RNNG is not simply due to the additional depth afforded by the tree LSTM composition function (a left branching tree, which always performs REDUCE when possible, is the “deepest” model). Under our RNNG parameterization, the right branching baseline is equivalent to the standard RNNLM and hence performs the same (we show the perplexity numbers as a sanity check). We found PRPN with default hyperparameters (which obtains a perplexity of 62.0 in the PTB setup from Mikolov et al. (2010)) to not perform well, but tuning hyperparameters improves performance.¹¹

The supervised RNNG performs well as a language model, despite being trained on the joint (rather than marginal) likelihood objective; the supervised RNNG is trained to maximize $\log p(x, \mathbf{z}; \theta)$ while the unsupervised RNNG is trained to maximize (a lower bound on) the language modeling objective $\log p(x; \theta)$. This indicates that explicit modeling of syntax helps generalization even with richly-parameterized

¹¹Using the code from <https://github.com/yikangshen/PRPN>, we tuned model size, initialization, dropout, learning rate, and use of batch normalization.

PTB	PPL
KN 5-gram (Dyer et al., 2016)	169.3
RNNLM (Dyer et al., 2016)	113.4
Original RNNG (Dyer et al., 2016)	102.4
Gated-Attention RNNG (Kuncoro et al., 2017)	100.9
Generative Dependency Parser (Buys & Blunsom, 2015)	138.6
Supervised Syntactic NLM (Buys & Blunsom, 2018)	107.6
Unsupervised Syntactic NLM (Buys & Blunsom, 2018)	125.2
PRPN [†] (Shen et al., 2018b)	96.7
Our work:	
RNNLM	93.2
URNNG	90.6
RNNG	88.7
RNNG \rightarrow URNNG	85.9
1M Sentences	PPL
PRPN [†] (Shen et al., 2018b)	77.7
RNNLM	77.4
URNNG	71.8
RNNG [‡]	72.9
RNNG [‡] \rightarrow URNNG	72.0

Table 5.2: (Top) Comparison of our work as a language model against prior works on sentence-level PTB with preprocessing from Dyer et al. (2016). Note that previous versions of RNNG differ from ours in terms of parameterization and model size. (Bottom) Results on a subset (1M sentences) of the one billion word corpus. PRPN[†] is the model from Shen et al. (2018b), whose hyperparameters were tuned by us. RNNG[‡] is trained on predicted parse trees from the self-attentive parser from Kitaev & Klein (2018).

neural models. Encouraged by these observations, we also experiment with a hybrid approach where we train a supervised RNNG first and continue fine-tuning the model (including the inference network) on the URNNG objective (RNNG \rightarrow URNNG in Table 5.1). Concretely, after training the supervised RNNG we fine-tune for 10 epochs with the unsupervised objective and use a smaller learning rate of 0.1 for the generative model. This approach results in nontrivial perplexity improvements, and suggests that it is possible to further improve language models with supervision on parsed

data, on top of the benefits afforded by the usual supervised RNNG training.

In Figure 5.2 we show perplexity by sentence length. We find that a standard language model (RNNLM) is better at modeling short sentences, but underperforms models that explicitly take into account structure (RNNG/URNNG) when the sentence length is greater than 10. Table 5.2 (top) compares our results against prior work on this version of the PTB, and Table 5.2 (bottom) shows the results on a 1M sentence subset of the one billion word corpus, which is two orders of magnitude larger than PTB. On this larger dataset URNNG still improves upon the RNNLM. We also trained an RNNG (and RNNG \rightarrow URNNG) on this dataset by parsing the training set with the self-attentive parser from Kitaev & Klein (2018), which obtains an F_1 score of 95.17 on the PTB test set.¹² These models improve upon the RNNLM but not the URNNG, potentially highlighting the limitations of using predicted trees for supervising RNNGs.

UNSUPERVISED PARSING Table 5.1 also shows the F_1 scores for unsupervised parsing, where we induce latent trees directly from words on the full dataset.¹³ For RNNG and URNNG we obtain the highest scoring tree from $q(\mathbf{z} | x; \phi)$ through the Viterbi version of the inside algorithm (i.e. CKY algorithm). Note that we could alternatively estimate

$$\arg \max_{\mathbf{z}} p(\mathbf{z} | x; \theta)$$

¹²We use the `benepar_en2` model from <https://github.com/nikitakit/self-attentive-parser>.

¹³Past work on grammar induction generally train/evaluate on short sentences and also assume access to gold POS tags (Klein & Manning, 2002; Smith & Eisner, 2004; Bod, 2006). However more recent works train directly on words (Jin et al., 2018b; Shen et al., 2018b; Drozdov et al., 2019).

Tree	PTB	CTB	Label	URNNG	PRPN
Gold	40.7	29.1	SBAR	74.8%	28.9%
Left	9.2	8.4	NP	39.5%	63.9%
Right	68.3	51.2	VP	76.6%	27.3%
Self	92.3	87.3	PP	55.8%	55.1%
RNNG	55.4	47.1	ADJP	33.9%	42.5%
PRPN	41.0	47.2	ADVP	50.4%	45.1%

Table 5.3: (Left) F_1 scores of URNNG against other trees. “Self” refers to another URNNG trained with a different random seed. (Right) Recall of constituents by label for URNNG and PRPN. Recall for a particular label is the fraction of ground truth constituents of that label that were identified by the model (as in [Htut et al. \(2018\)](#)).

by sampling parse trees from $q(\mathbf{z} | x; \phi)$ and using $p(x, \mathbf{z}; \theta)$ to rerank the output, as in [Dyer et al. \(2016\)](#); however we found this variational approximation to work well in practice. We calculate unlabeled F_1 using `evalb`, which ignores punctuation and discards trivial spans (width-one and sentence spans).¹⁴ Since we compare F_1 against the original, non-binarized trees per convention in unsupervised parsing, F_1 scores of models using oracle binarized trees constitute the upper bounds. We confirm the replication study of [Htut et al. \(2018\)](#) and find that PRPN is a strong model for grammar induction. URNNG performs on par with PRPN on English but PRPN does better on Chinese; both outperform right branching baselines. Table 5.3 further analyzes the learned trees and shows the F_1 score of URNNG trees against other trees (left), and the recall of URNNG/PRPN trees against ground truth constituents (right). We find that trees induced by URNNG and PRPN are quite different; URNNG is more sensitive to SBAR and VP, while PRPN is better at identifying NP. While left as future work, this naturally suggests a hybrid approach wherein the intersection of constituents from URNNG and PRPN is used to create a corpus of

¹⁴Available at <https://nlp.cs.nyu.edu/evalb/>. We evaluate with `COLLINS.prm` parameter file and `LABELED` option equal to 0.

partially annotated trees, which can be used to guide another model, e.g. via semi-supervision (Hwa, 1999).

5.4.3 ANALYSIS

DISTRIBUTIONAL METRICS Table 5.4 shows some standard metrics related to the learned generative model/inference network, similar to those explored in chapter 3. The “reconstruction” perplexity based on

$$\mathbb{E}_{q(\mathbf{z} | x; \phi)}[\log p(x | \mathbf{z}; \theta)],$$

is much lower than actual perplexity, and further, the Kullback-Leibler divergence between the conditional prior and the variational posterior, given by

$$\mathbb{E}_{q(\mathbf{z} | x; \phi)} \left[\log \frac{q(\mathbf{z} | x; \phi)}{p(\mathbf{z} | x_{<\mathbf{z}}; \theta)} \right],$$

is highly nonzero. This indicates that the latent space is being used in a meaningful way and that there is no posterior collapse (Bowman et al., 2016). As expected, the entropy of the variational posterior is much lower than the entropy of the conditional prior, but there is still some uncertainty in the posterior.

SYNTACTIC EVALUATION We perform a syntactic evaluation of the different models based on the setup from Marvin & Linzen (2018): the model is given two minimally different sentences, one grammatical and one ungrammatical, and must identify the

	PTB		CTB	
	RNNG	URNNG	RNNG	URNNG
PPL	88.7	90.6	193.1	195.7
Recon. PPL	74.6	73.4	183.4	151.9
KL	7.10	6.13	11.11	8.91
Prior Entropy	7.65	9.61	9.48	15.13
Post. Entropy	1.56	2.28	6.23	5.75
Unif. Entropy	26.07	26.07	30.17	30.17

Table 5.4: Metrics related to the generative model/inference network for RNNG/URNNG. For the supervised RNNG we take the “inference network” to be the discriminative parser trained alongside the generative model. Recon. PPL is the reconstruction perplexity based on $\mathbb{E}_{q(\mathbf{z} | x; \phi)} [\log p(x | \mathbf{z}; \theta)]$, and KL is the Kullback-Leibler divergence. Prior entropy is the entropy of the conditional prior $p(\mathbf{z} | x_{<\mathbf{z}}; \theta)$, and uniform entropy is the entropy of the uniform distribution over all binary trees. The KL/entropy metrics are averaged across sentences.

grammatical sentence by assigning it higher probability, e.g.

the senators near the assistant are old

*the senators near the assistant is old

We modify the publicly available dataset from https://github.com/BeckyMarvin/LM_syneval to only keep sentence pairs that did not have any unknown words with respect to our vocabulary, resulting in 80K sentence pairs for evaluation.¹⁵ Table 5.5 shows the accuracy results. Overall the supervised RNNG significantly outperforms the other models, indicating opportunities for further work in unsupervised modeling. While the URNNG does slightly outperform an RNNLM, the distribution of errors made from both models are similar, and thus it is not clear whether the outperformance is simply due to better perplexity or learning different structural biases.

¹⁵Further, we train on a much smaller corpus, and hence our results are not directly comparable to the numbers reported in [Marvin & Linzen \(2018\)](#)

	RNNLM	PRPN	RNNG	URNNG
PPL	93.2	96.7	88.7	90.6
Overall	62.5%	61.9%	69.3%	64.6%
Subj.	63.5%	63.7%	89.4%	67.2%
Obj. Rel.	62.6%	61.0%	67.6%	65.7%
Refl.	60.7%	68.8%	57.3%	60.5%
NPI	58.7%	39.5%	46.8%	55.0%

Table 5.5: Syntactic evaluation based on the setup from [Marvin & Linzen \(2018\)](#). Subj. is subject-verb agreement in sentential complement, across prepositional phrase/subjective relative clause, and VP coordination; Obj. Rel. refers to subject-verb agreement in/across an objective relative clause; Refl. refers to reflexive pronoun agreement with antecedent; NPI is negative polarity items.

5.5 DISCUSSION

5.5.1 LIMITATIONS

There are several limitations to our approach. For one, the URNNG takes considerably more time/memory to train than a standard language model due to the $O(T^3)$ dynamic program in the inference network, multiple samples to obtain control variates for variance reducing gradient estimators, and dynamic computation graphs that make efficient batching nontrivial.¹⁶ We also found the inference network to be sensitive to the parameterization of span scores (i.e. it was important to use the “subtraction” parameterization from [Wang & Chang \(2016\)](#)), which indicates that the context-freeness of the variational family is by itself not enough to provide the right inductive bias. The framework is also sensitive to other hyperparameters and required various optimization strategies (e.g. separate optimizers for the inference network and

¹⁶The main time bottleneck is the dynamic computation graph, since the dynamic programming algorithm can be batched (however the latter is a significant memory bottleneck). We manually batch the SHIFT and REDUCE operation as much as possible, though recent work on auto-batching ([Neubig et al., 2017](#)) could potentially make this easier/faster.

the generative model, annealing the multiplier on the KL portion of the objective) to avoid posterior collapse.

In our experiments we adopt the experimental setup and preprocessed dataset from the original RNNG work (Dyer et al., 2016) (which is itself from the standard supervised parsing setup), which does not discard punctuation. However, in subsequent experiment we were unable to improve upon a right-branching baseline when training the URNNG on a version of PTB where punctuation is removed.¹⁷ Syntax and punctuation are related (Nunberg, 1990), and its inclusion in learning unsupervised parsing systems can be justified by the fact that punctuation can be partially seen as an analog to prosody in speech. However, an unsupervised parser should ideally still work well even when trained on corpora without punctuation. In the next chapter we revisit unsupervised parsing and propose an alternative approach which is able to learn nontrivial trees from unpunctuated data.

5.5.2 RICH GENERATIVE MODELS FOR LEARNING LATENT STRUCTURES

Despite the fact that rich generative models—such as autoregressive models that fully condition on their history such as RNNs or RNNGs—can model the underlying data well in terms of perplexity, in chapter 3 we questioned the prudence of utilizing them for learning meaningful latent variables, given the fact that if the generative model is powerful enough it can model $p_{\star}(x)$ without utilizing the latent variable. We saw in chapter 3 that one way to encourage meaningful use of the latent space is to reduce the amortization gap by better optimizing the variational posterior distribution.

¹⁷Many prior works that induce trees directly from words often employ additional heuristics based on punctuation (Seginer, 2007; Ponvert et al., 2011; Spitkovsky et al., 2013; Parikh et al., 2014), as punctuation (e.g. comma) is usually a reliable signal for start/end of constituent spans. We also reiterate that punctuation is used during training but ignored during evaluation.

In this chapter, we explored an alternative method to encourage meaningful latent space via posterior regularization (Ganchev et al., 2010), where we employ a discriminative, context-free parser to imbue the generative model with the appropriate inductive bias. In particular we exploit the ELBO objective to encourage the learning of generative models whose true posterior is close to the variational family of context-free distributions over trees. We found the use of this structured inference network to be crucial in learning linguistically meaningful parse trees.

Note that our use of posterior regularization via independence assumptions on the variational family is quite a weak form of constraint. An interesting future direction would involve richer inductive biases from posterior regularization by further constraining the inference network, for example based on universal syntactic rules (Naseem et al., 2010). Alternatively, we could add auxiliary objectives such as the contrastive loss proposed by Smith & Eisner (2005) to better train the inference network. Or further still, we could utilize a PCFG as an inference network which not only encodes context-free assumptions but also allows for tractable training of the inference network itself on the corpus log marginal likelihood. We explore some of these extensions further in the next chapter, where we pretrain the inference network on parse trees obtained from an induced probabilistic grammar.

5.6 RELATED WORK

There has been much work on incorporating tree structures into statistical models for syntax-aware language modeling, both for unconditional (Emami & Jelinek, 2005; Buys & Blunsom, 2015; Dyer et al., 2016) and conditional (Yin & Neubig, 2017; Alvarez-Melis & Jaakkola, 2017b; Rabinovich et al., 2017; Aharoni & Goldberg, 2017;

Eriguchi et al., 2017; Wang et al., 2018b; Gū et al., 2018) cases. These approaches generally rely on annotated parse trees during training and maximize the joint likelihood of sentence-tree pairs. Neural approaches to combining language modeling and unsupervised tree learning typically embed soft, tree-like structures as hidden layers of a deep network (Cho et al., 2014a; Chung et al., 2017; Shen et al., 2018b, 2019). In contrast, Buys & Blunsom (2018) make Markov assumptions and perform exact marginalization over latent dependency trees. Our work is also related to the recent line of work on learning latent trees as part of a deep model through supervision on other tasks, typically via differentiable structured hidden layers (Kim et al., 2017; Bradbury & Socher, 2017; Liu & Lapata, 2017; Tran & Bisk, 2018; Peng et al., 2018; Niculae et al., 2018b; Liu et al., 2018), policy gradient-based approaches (Yogatama et al., 2017; Williams et al., 2018; Havrylov et al., 2019), or differentiable relaxations (Choi et al., 2018; Maillard & Clark, 2018).

The variational approximation uses amortized inference (Kingma & Welling, 2014; Mnih & Gregor, 2014; Rezende et al., 2014), in which an inference network is used to obtain the variational posterior for each observed x . Since our inference network is structured (i.e., a CRF), it is also related to CRF autoencoders (Ammar et al., 2014) and structured VAEs (Johnson et al., 2016; Krishnan et al., 2018), which have been used previously for unsupervised (Cai et al., 2017; Drozdov et al., 2019; Li et al., 2019a) and semi-supervised (Yin et al., 2018; Corro & Titov, 2018) parsing.

5.7 CONCLUSION

It is an open question as to whether explicit modeling of syntax significantly helps neural models. Strubell et al. (2018) find that supervising intermediate attention lay-

ers with syntactic heads improves semantic role labeling, while [Shi et al. \(2018\)](#) observe that for text classification, syntactic trees only have marginal impact. In this chapter, we show that at least for language modeling, incorporating syntax either via explicit supervision or as latent variables does provide useful inductive biases and improves performance. Our results, along with other recent work on joint language modeling/structure learning with deep networks ([Shen et al., 2018b, 2019](#); [Wiseman et al., 2018](#); [Kawakami et al., 2018](#)), suggest that it is possible to learn generative models of language that model the underlying data well (i.e. assign high likelihood to held-out data) and at the same time induce meaningful linguistic structures.

6

Latent Variable Model of Grammars & Collapsed Variational Inference

6.1 INTRODUCTION

In this chapter we consider the problem of grammar induction with deep latent variable models. Unlike in chapter 5 where we were interested in learning a generative

The material in this chapter is adapted from [Kim et al. \(2019a\)](#).

model that has low perplexity *and* performs well as an unsupervised parser, in this chapter we mainly focus on the case where the underlying structure is itself the goal. This will, as we will shortly see, allow us to work with generative models that have better inductive bias towards meaningful structure and at the same time admit tractable inference algorithms through partial conditioning.

Learning generative grammars (i.e., recursive rewrite rules that generate language) from observed yields is a classical problem in natural language processing and computational linguistics, with historical roots going all the way back to Chomsky (1957):

One may arrive at a grammar by intuition, guess-work, all sorts of partial methodological hints, reliance on past experience, etc. It is no doubt possible to give an organized account of many useful procedures of analysis, but it is questionable whether these can be formulated rigorously, exhaustively and simply enough to qualify as a practical and mechanical discovery algorithm.

Initial theoretical results on grammar induction was largely negative. Gold’s Theorem showed that it is not possible to learn even regular grammars from positive examples alone (Gold, 1967). However, the notion of learnability in the aforementioned work is rather strict and roughly requires that the learner is able to correctly identify the language after a finite number of mistakes (*identification in the limit*). Employing a different notion of learnability (*measure-one learnability*), Horning (1969) showed that it is possible to learn *probabilistic* grammars from positive examples alone, though the algorithm given in Horning (1969) is enumerative and thus not computationally practical.

Probabilistic approaches to grammar induction generally require specifying a probabilistic grammar (e.g. formalism, number and shape of rules), and fitting its parameters through optimization. Early work found that it was difficult to induce probabilis-

tic context-free grammars (PCFG) from natural language data through direct methods, such as optimizing the log likelihood with the EM algorithm (Carroll & Charniak, 1992; Charniak, 1993). While the reasons for the failure are manifold and not completely understood, two major potential causes are the ill-behaved optimization landscape and the overly strict independence assumptions of PCFGs. More successful approaches to grammar induction have thus resorted to carefully-crafted auxiliary objectives (Klein & Manning, 2002), priors or non-parametric models (Kurihara & Sato, 2006; Johnson et al., 2007; Liang et al., 2007; Wang & Blunsom, 2013), and manually-engineered features (Huang et al., 2012; Golland et al., 2012) to encourage the desired structures to emerge.

In this chapter, we revisit these aforementioned issues in light of advances in deep parameterization and inference. Contrary to common wisdom, we find that parameterizing a PCFG’s rule probabilities with neural networks over distributed representations makes it possible to induce linguistically meaningful grammars by simple maximum likelihood learning. While the optimization problem remains non-convex, recent work suggests that there are optimization benefits afforded by over-parameterized models (Arora et al., 2018; Xu et al., 2018; Du et al., 2019), and we indeed find that this neural PCFG is significantly easier to optimize than the traditional PCFG. This factored parameterization makes it straightforward to incorporate side information into rule probabilities through a sentence-level continuous latent vector, which effectively allows different contexts in a derivation to coordinate. In this compound PCFG—continuous mixture of PCFGs—the context-free assumptions hold conditioned on the latent vector but not unconditionally, thereby obtaining longer-range dependencies within a tree-based generative process. While compound PCFGs break efficient exact inference, if the latent vector is known the distribution over trees re-

duces to a standard PCFG. This property allows us to perform learning and inference with collapsed variational inference where the latent trees are marginalized out exactly with dynamic programming, and standard amortized inference is used to handle the latent vector.

Evaluating generative grammars is itself a difficult problem, and a standard approach is to use the grammar to parse a set of sentences and compare the predicted trees against the linguistically annotated trees. That is, we evaluate the learned grammar as an unsupervised parsing system. On unsupervised parsing benchmarks for English and Chinese, the proposed approach is found to perform favorably against recent neural network-based approaches to unsupervised parsing, including the unsupervised recurrent neural network grammar studied in the previous section. Finally, we show that the induced trees can be used to supervise a recurrent neural network grammar that performs well as a language model and at the same time learns meaningful linguistic structure.

6.2 BACKGROUND

6.2.1 PROBABILISTIC CONTEXT-FREE GRAMMARS

We consider context-free grammars (CFG) consisting of a 5-tuple $\mathcal{G} = (S, \mathcal{N}, \mathcal{P}, \Sigma, \mathcal{R})$ where S is the distinguished start symbol, \mathcal{N} is a finite set of nonterminals, \mathcal{P} is a finite set of preterminals, Σ is a finite set of terminal symbols, and \mathcal{R} is a finite set of

rules of the form,

$$\begin{aligned}
S &\rightarrow A, & A &\in \mathcal{N}, \\
A &\rightarrow B \ C, & A &\in \mathcal{N}, \ B, C \in \mathcal{N} \cup \mathcal{P}, \\
T &\rightarrow w, & T &\in \mathcal{P}, \ w \in \Sigma.
\end{aligned}$$

A *probabilistic* context-free grammar (PCFG) consists of a grammar \mathcal{G} and rule probabilities $\boldsymbol{\pi} = \{\pi_r\}_{r \in \mathcal{R}}$ such that π_r is the probability of the rule r . Since we will be inducing a grammar directly from words, \mathcal{P} will roughly correspond to part-of-speech tags and \mathcal{N} will correspond to constituent labels.

Letting $\mathcal{T}_{\mathcal{G}}$ be the set of all parse trees of \mathcal{G} , a PCFG defines a probability distribution over trees $\mathbf{t} \in \mathcal{T}_{\mathcal{G}}$ via

$$p(\mathbf{t}; \boldsymbol{\pi}) = \prod_{r \in \mathbf{t}_{\mathcal{R}}} \pi_r,$$

where $\mathbf{t}_{\mathcal{R}}$ is the set of rules used in the derivation of \mathbf{t} . It also defines a distribution over string of terminals $x \in \Sigma^*$ via

$$p(x) = \sum_{\mathbf{t} \in \mathcal{T}_{\mathcal{G}}(x)} p(\mathbf{t}; \boldsymbol{\pi}),$$

where $\mathcal{T}_{\mathcal{G}}(x) = \{\mathbf{t} \mid \text{yield}(\mathbf{t}) = x\}$, i.e. the set of trees \mathbf{t} such that \mathbf{t} 's leaves are x . We will slightly abuse notation and use

$$p(\mathbf{t} \mid x; \boldsymbol{\pi}) \triangleq \frac{\mathbb{1}[\text{yield}(\mathbf{t}) = x] p(\mathbf{t}; \boldsymbol{\pi})}{p(x; \boldsymbol{\pi})}$$

to denote the posterior distribution over the unobserved latent trees given the ob-

served sentence x , where $\mathbb{1}[\cdot]$ is the indicator function.¹ For a given sentence x , we can efficiently perform posterior inference and marginalization over the exponentially-sized set $\mathcal{T}_{\mathcal{G}}(x)$ using dynamic programming (Baker, 1979).

6.2.2 GRAMMAR INDUCTION VS. UNSUPERVISED PARSING

In this chapter we focus on learning a formal grammar, i.e., a set of recursive rewrite rules and their associated probabilities which give a stochastic account of how sentences in natural language are generated. Once learned, we will evaluate the induced grammar on *unsupervised parsing*. That is, we will use the induced grammar to parse a set of unseen test sentences and compare the predicted parse trees against linguistically-annotated parse trees. It is worth noting that grammar induction and unsupervised parsing are not synonymous. Indeed, there is much prior work focusing on learning an unsupervised parser without learning an underlying grammar (Klein & Manning, 2002; Smith & Eisner, 2005; Bod, 2006; Seginer, 2007; Shen et al., 2018b, 2019). However, for the purposes of this chapter we will not dwell on the distinction too much since we will still evaluate the induced grammar as an unsupervised parser.

6.3 COMPOUND PROBABILISTIC CONTEXT-FREE GRAMMARS

In this section we first describe a *neural* PCFG that makes use of neural networks over symbol embeddings to obtain rule probabilities. We then show that this embedding parameterization admits a natural extension to a richer grammar that makes use of a compound generative process. This *compound* PCFG models long-range dependencies through a sentence-level latent vector. Finally, we describe a collapsed

¹Therefore when used in the context of posterior distributions $p(\mathbf{t} | x)$, \mathbf{t} does not include the leaves x and only refers to the unobserved latent nonterminal/preterminal symbols.

amortized variational inference approach for performing inference in the compound PCFG.

6.3.1 A NEURAL PARAMETERIZATION

The standard way to parameterize a PCFG is to simply associate a scalar to each rule π_r with the constraint that they form valid probability distributions, i.e. each nonterminal is associated with a fully-parameterized categorical distribution over its possible rules. This *direct parameterization* is algorithmically convenient since the M-step in the EM algorithm (Dempster et al., 1977) has a closed form. However, there is a long history of work showing that it is difficult to learn meaningful grammars from natural language data with this parameterization (Carroll & Charniak, 1992). Successful approaches to unsupervised parsing have therefore modified the model/learning objective by guiding potentially unrelated rules to behave similarly, through, for example, prior distributions over the categorical distributions.

Recognizing that sharing among rule types is beneficial, in this chapter we consider a *neural parameterization* where rule probabilities are based on distributed representations of symbols within the grammar. We associate embeddings with each symbol, introducing input embeddings \mathbf{w}_N for each symbol N on the left side of a rule i.e.

$$N \in \{S\} \cup \mathcal{N} \cup \mathcal{P}.$$

For each rule type r , π_r is parameterized as follows,

$$\begin{aligned}\pi_{S \rightarrow A} &= \frac{\exp(\mathbf{u}_A^\top \text{MLP}_1(\mathbf{w}_S; \theta)) + b_A}{\sum_{A' \in \mathcal{N}} \exp(\mathbf{u}_{A'}^\top \text{MLP}_1(\mathbf{w}_S; \theta) + b_{A'})}, \\ \pi_{A \rightarrow BC} &= \frac{\exp(\mathbf{u}_{BC}^\top \mathbf{w}_A + b_{BC})}{\sum_{B'C' \in \mathcal{M}} \exp(\mathbf{u}_{B'C'}^\top \mathbf{w}_A + b_{B'C'})}, \\ \pi_{T \rightarrow w} &= \frac{\exp(\mathbf{u}_w^\top \text{MLP}_2(\mathbf{w}_T; \theta) + b_w)}{\sum_{w' \in \Sigma} \exp(\mathbf{u}_{w'}^\top \text{MLP}_2(\mathbf{w}_T; \theta) + b_{w'})},\end{aligned}$$

where in the second equation the softmax normalizes over the product space

$$\mathcal{M} = (\mathcal{N} \cup \mathcal{P}) \times (\mathcal{N} \cup \mathcal{P}).$$

The MLPs have two residual layers,

$$\begin{aligned}\text{MLP}(\mathbf{x}) &= \text{ResidualLayer}_1(\text{ResidualLayer}_2(\mathbf{W}\mathbf{x} + \mathbf{b})), \\ \text{ResidualLayer}(\mathbf{y}) &= \text{ReLU}(\mathbf{V} \text{ReLU}(\mathbf{U}\mathbf{y} + \mathbf{p}) + \mathbf{q}) + \mathbf{y}.\end{aligned}$$

We have separate MLPs for $\pi_{S \rightarrow A}$ and $\pi_{T \rightarrow w}$, and do not employ an MLP for $\pi_{A \rightarrow BC}$ as we found the model to perform better without an MLP for rules of this form. We will use

$$\mathbf{E}_{\mathcal{G}} = \{\mathbf{w}_N \mid N \in \{S\} \cup \mathcal{N} \cup \mathcal{P}\} \cup \{\mathbf{u}_M \mid M \in \mathcal{N} \cup \mathcal{M} \cup \Sigma\}$$

to denote the set of input/output symbol embeddings for a grammar \mathcal{G} , and λ to refer to the parameters of the neural network used to obtain the rule probabilities (i.e. the parameters of the MLPs). A graphical model-like illustration of the neural PCFG is shown in Figure 6.1 (left). It is clear that the neural parameterization does not change the underlying context-free assumptions. The difference between the two

is analogous to the difference between count-based vs. feed-forward neural language models, where feed-forward neural language models make the same Markov assumptions as the count-based models but are able to take advantage of shared, distributed representations that promote parameter sharing.

6.3.2 A COMPOUND EXTENSION

A compound probability distribution (Robbins, 1951) is a distribution whose parameters are themselves random variables. These distributions generalize mixture models to the continuous case, for example in factor analysis which assumes the following generative process,

$$\mathbf{z} \sim \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), \quad \mathbf{x} \sim \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{z}, \mathbf{\Sigma}).$$

Compound distributions provide the ability to model rich generative processes, but marginalizing over the latent parameter can be computationally intractable unless conjugacy can be exploited.

In this chapter we study compound probabilistic context-free grammars whose distribution over trees arises from the following generative process: we first obtain rule probabilities via

$$\mathbf{z} \sim p(\mathbf{z}; \gamma) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), \quad \boldsymbol{\pi}_{\mathbf{z}} = \text{NeuralNet}(\mathbf{z}, \mathbf{E}_{\mathcal{G}}; \lambda),$$

where $p(\mathbf{z}; \gamma)$ is a prior with parameters γ (spherical Gaussian in this chapter), and $\text{NeuralNet}(\mathbf{z}, \mathbf{E}_{\mathcal{G}}; \lambda)$ is a neural network that concatenates the input symbol embed-

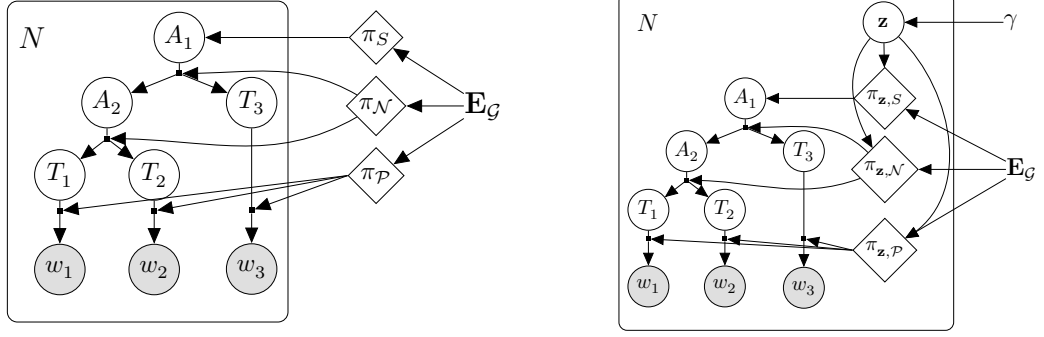


Figure 6.1: A graphical model-like diagram for the neural PCFG (left) and the compound PCFG (right) for an example tree structure. In the above, $A_1, A_2 \in \mathcal{N}$ are nonterminals, $T_1, T_2, T_3 \in \mathcal{P}$ are preterminals, $w_1, w_2, w_3 \in \Sigma$ are terminals. In the neural PCFG, the global rule probabilities $\pi = \pi_S \cup \pi_N \cup \pi_P$ are the output from a neural net run over the symbol embeddings \mathbf{E}_G , where π_N are the set of rules with a nonterminal on the left hand side (π_S and π_P are similarly defined). In the compound PCFG, we have per-sentence rule probabilities $\pi_{\mathbf{z}} = \pi_{\mathbf{z},S} \cup \pi_{\mathbf{z},N} \cup \pi_{\mathbf{z},P}$ obtained from running a neural net over a random vector \mathbf{z} (which varies across sentences) and global symbol embeddings \mathbf{E}_G . In this case, the context-free assumptions hold conditioned on \mathbf{z} , but they do not hold unconditionally: e.g. when conditioned on \mathbf{z} and A_2 , the variables A_1 and T_1 are independent; however when conditioned on just A_2 , they are not independent due to the dependence path through \mathbf{z} . Note that the rule probabilities are random variables in the compound PCFG but deterministic variables in the neural PCFG.

dings with \mathbf{z} and outputs the sentence-level rule probabilities $\pi_{\mathbf{z}}$,

$$\begin{aligned} \pi_{\mathbf{z}, S \rightarrow A} &= \frac{\exp(\mathbf{u}_A^\top \text{MLP}_1([\mathbf{w}_S; \mathbf{z}]; \theta)) + b_A}{\sum_{A' \in \mathcal{N}} \exp(\mathbf{u}_{A'}^\top \text{MLP}_1([\mathbf{w}_S; \mathbf{z}]; \theta) + b_{A'})}, \\ \pi_{\mathbf{z}, A \rightarrow BC} &= \frac{\exp(\mathbf{u}_{BC}^\top [\mathbf{w}_A; \mathbf{z}] + b_{BC})}{\sum_{B'C' \in \mathcal{M}} \exp(\mathbf{u}_{B'C'}^\top [\mathbf{w}_A; \mathbf{z}] + b_{B'C'})}, \\ \pi_{\mathbf{z}, T \rightarrow w} &= \frac{\exp(\mathbf{u}_w^\top \text{MLP}_2([\mathbf{w}_T; \mathbf{z}]; \theta) + b_w)}{\sum_{w' \in \Sigma} \exp(\mathbf{u}_{w'}^\top \text{MLP}_2([\mathbf{w}_T; \mathbf{z}]; \theta) + b_{w'})}, \end{aligned}$$

The MLPs are as in the neural PCFG where the first layer’s input dimensions are appropriately changed to account for concatenation with \mathbf{z} . Then a tree/sentence is sampled from a PCFG with rule probabilities given by $\pi_{\mathbf{z}}$,

$$\mathbf{t} \sim \text{PCFG}(\pi_{\mathbf{z}}), \quad x = \text{yield}(\mathbf{t}).$$

This can be viewed as a continuous mixture of PCFGs, or alternatively, a Bayesian PCFG with a prior on sentence-level rule probabilities parameterized by $\mathbf{z}, \lambda, \mathbf{E}_G$.² Importantly, under this generative model the context-free assumptions hold *conditioned on \mathbf{z}* , but they do not hold unconditionally. This is shown in Figure 6.1 (right) where there is a dependence path through \mathbf{z} if it is not conditioned upon. Compound PCFGs give rise to a marginal distribution over parse trees \mathbf{t} via

$$p(\mathbf{t}; \theta) = \int p(\mathbf{t} | \mathbf{z}; \theta) p(\mathbf{z}; \gamma) d\mathbf{z},$$

where $p(\mathbf{t} | \mathbf{z}; \theta) = \prod_{r \in \mathbf{t}_{\mathcal{R}}} \pi_{\mathbf{z}, r}$. The subscript in $\pi_{\mathbf{z}, r}$ denotes the fact that the rule probabilities depend on \mathbf{z} . Compound PCFGs are clearly more expressive than PCFGs as each sentence has its own set of rule probabilities. However, it still assumes a tree-based generative process, making it possible to learn latent tree structures.

One motivation for the compound PCFG is the fact that the simple, unlexicalized context-free grammars such as the one that we have been working with are unlikely to be adequate statistical models of natural language, even though their simplicity facilitates efficient training.³ We can in principle model richer dependencies through vertical/horizontal Markovization (Johnson, 1998; Klein & Manning, 2003) and lexicalization (Collins, 1997). However such dependencies complicate training due to the rapid increase in the number of rules. Under this view, we can interpret the com-

²Under the Bayesian PCFG view, $p(\mathbf{z}; \gamma)$ is a distribution over \mathbf{z} (a subset of the prior), and is thus a hyperprior.

³A piece of evidence for the misspecification of first-order, unlexicalized PCFGs as a statistical model of natural language is that if one pretrains them on supervised data and continues training with the unsupervised objective (i.e. log marginal likelihood), the resulting grammar deviates significantly from the supervised initial grammar while the log marginal likelihood improves (Johnson et al., 2007). Similar observations have been made for part-of-speech induction with Hidden Markov Models (Merialdo, 1994).

pound PCFG as a restricted version of some lexicalized, higher-order PCFG where a child can depend on structural and lexical context through a shared latent vector.⁴ We hypothesize that this dependence among siblings is especially useful in grammar induction from words, where (for example) if we know that `watched` is used as a verb then the noun phrase is likely to be `a movie`.

In contrast to the usual Bayesian treatment of PCFGs which places priors on global rule probabilities (Kurihara & Sato, 2006; Johnson et al., 2007; Wang & Blunsom, 2013), the compound PCFG assumes a prior on local, sentence-level rule probabilities. It is therefore closely related to the Bayesian grammars studied by Cohen et al. (2009) and Cohen & Smith (2009), who also sample local rule probabilities from a logistic normal prior for training dependency models with valence (DMV) (Klein & Manning, 2004).

6.3.3 TRAINING AND INFERENCE

The expressivity of compound PCFGs comes at a significant challenge in learning and inference. Letting θ be the parameters of the generative model, we would like to maximize the log marginal likelihood of the observed sentence $\log p(x; \theta)$. In the neural PCFG the log marginal likelihood

$$\log p(x; \theta) = \log \sum_{\mathbf{t} \in \mathcal{T}_G(x)} p(\mathbf{t}; \theta),$$

⁴Another interpretation of the compound PCFG is to view it as a vectorized version of *indexed* grammars (Aho, 1968), which extend CFGs by augmenting nonterminals with additional index strings that may be inherited or modified during derivation. Compound PCFGs instead equip nonterminals with a continuous vector that is always inherited.

can be obtained by summing out the latent tree structure using the inside algorithm (Baker, 1979), which is differentiable and thus amenable to gradient-based optimization. In the compound PCFG, the log marginal likelihood is given by,

$$\begin{aligned}\log p(x; \theta) &= \log \left(\int p(x | \mathbf{z}; \theta) p(\mathbf{z}; \gamma) d\mathbf{z} \right) \\ &= \log \left(\int \sum_{\mathbf{t} \in \mathcal{T}_G(x)} p(\mathbf{t} | \mathbf{z}; \theta) p(\mathbf{z}; \gamma) d\mathbf{z} \right).\end{aligned}$$

Notice that while the integral over \mathbf{z} makes this quantity intractable, when we condition on \mathbf{z} , we can tractably perform the inner summation to obtain $p(x | \mathbf{z}; \theta)$ using the inside algorithm. We therefore resort to collapsed amortized variational inference, where we first obtain a sample \mathbf{z} from a variational posterior distribution (given by an amortized inference network with parameters ϕ), then perform the inner marginalization conditioned on this sample. As usual, the evidence lower bound is given by,

$$\text{ELBO}(\theta, \phi; x) = \mathbb{E}_{q(\mathbf{z} | x; \phi)}[\log p(x | \mathbf{z}; \theta)] - \text{KL}[q(\mathbf{z} | x; \phi) \| p(\mathbf{z}; \gamma)],$$

and we can calculate $p(x | \mathbf{z}; \theta)$ given a sample \mathbf{z} from a variational posterior $q(\mathbf{z} | x; \phi)$. For the variational family we use a diagonal Gaussian where the mean/log-variance vectors are given by an affine layer over max-pooled hidden states from an LSTM over x . We can then obtain low-variance estimators for $\nabla_{\theta, \phi} \text{ELBO}(\theta, \phi; x)$ by using the reparameterization trick for the expected reconstruction likelihood and the analytical expression for the KL term, as we saw in chapter 3. We remark that under the Bayesian PCFG view, since the parameters of the prior (i.e. θ) are estimated from the data, our approach can be seen as an instance of empirical Bayes (Robbins, 1956).

MAXIMUM A POSTERIORI INFERENCE After training, we are interested in comparing the learned trees against an annotated treebank. This requires inferring the most likely tree given a sentence, i.e.

$$\arg \max_{\mathbf{t}} p(\mathbf{t} | x; \theta).$$

For the neural PCFG we can obtain the most likely tree by using the Viterbi version of the inside algorithm (CKY algorithm). For the compound PCFG, the arg max is intractable to obtain exactly, and hence we estimate it with the following approximation,

$$\arg \max_{\mathbf{t}} \int p(\mathbf{t} | x, \mathbf{z}; \theta) p(\mathbf{z} | x; \theta) d\mathbf{z} \approx \arg \max_{\mathbf{t}} p(\mathbf{t} | x, \boldsymbol{\mu}_{\phi}(x); \theta),$$

where $\boldsymbol{\mu}_{\phi}(x)$ is the mean vector from the inference network. The above approximates the true posterior $p(\mathbf{z} | x; \theta)$ with $\delta(\mathbf{z} - \boldsymbol{\mu}_{\phi}(x))$, the Dirac delta function at the mode of the variational posterior.⁵ This quantity is tractable as in the PCFG case. Other approximations are possible: for example we could use $q(\mathbf{z} | x; \phi)$ as an importance sampling distribution to estimate the first integral. However we found the above approximation to be efficient and effective in practice.

⁵Since $p(\mathbf{t} | x, \mathbf{z}; \theta)$ is continuous with respect to \mathbf{z} , we have $\int p(\mathbf{t} | x, \mathbf{z}; \theta) \delta(\mathbf{z} - \boldsymbol{\mu}_{\phi}(x)) d\mathbf{z} = p(\mathbf{t} | x, \boldsymbol{\mu}_{\phi}(x); \theta)$.

6.4 EMPIRICAL STUDY

6.4.1 EXPERIMENTAL SETUP

DATA We test our approach on the English Penn Treebank (PTB) (Marcus et al., 1993) with the standard splits (2-21 for training, 22 for validation, 23 for test) and the same preprocessing as in Shen et al. (2018b) and Shen et al. (2019), where we discard punctuation, lowercase all tokens, and take the top 10K most frequent words as the vocabulary. This setup differs from the setup from chapter 5 since we discard punctuation and use a smaller vocabulary. We further experiment on Chinese with version 5.1 of the Chinese Penn Treebank (CTB) (Xue et al., 2005), with the same splits as in Chen & Manning (2014). On CTB we also remove punctuation and keep the top 10K word types.

HYPERPARAMETERS Our PCFG uses 30 nonterminals and 60 preterminals, with 256-dimensional symbol embeddings. The compound PCFG uses 64-dimensional latent vectors. The bidirectional LSTM inference network has a single layer with 512 dimensions, and the mean and the log variance vector for $q(\mathbf{z} | x; \phi)$ are given by max-pooling the hidden states of the LSTM and passing it through an affine layer. Model parameters are initialized with Xavier uniform initialization. For training we use Adam with $\beta_1 = 0.75$, $\beta_2 = 0.999$ and learning rate of 0.001, with a maximum gradient norm limit of 3. We train for 10 epochs with batch size equal to 4. We employ a curriculum learning strategy (Bengio et al., 2009) where we train only on sentences of length up to 30 in the first epoch, and increase this length limit by 1 each epoch. Similar curriculum-based strategies have been used in the past for grammar induction (Spitkovsky et al., 2012). During training we perform early stopping based on vali-

dation perplexity.⁶ Finally, to mitigate against overfitting to PTB, experiments on CTB utilize the same hyperparameters from PTB.

BASELINES We observe that even on PTB there is enough variation in setups across prior work on grammar induction to render a meaningful comparison difficult. Some important dimensions along which prior works vary include, (1) input data: earlier work on grammar induction generally assumed gold (or induced) part-of-speech tags (Klein & Manning, 2004; Smith & Eisner, 2004; Bod, 2006; Snyder et al., 2009), while more recent works induce grammar directly from words (Spitkovsky et al., 2013; Shen et al., 2018b); (2) use of punctuation: even within papers that induce a grammar directly from words, some papers employ heuristics based on punctuation as punctuation is usually a strong signal for start/end of constituents (Seginer, 2007; Ponvert et al., 2011; Spitkovsky et al., 2013), some train with punctuation (Jin et al., 2018b; Drozdov et al., 2019; Kim et al., 2019b), while others discard punctuation altogether for training (Shen et al., 2018b, 2019); (3) train/test data: some works do not explicitly separate out train/test sets (Reichart & Rappoport, 2010; Golland et al., 2012) while some do (Huang et al., 2012; Parikh et al., 2014; Htut et al., 2018). Maintaining train/test splits is less of an issue for unsupervised structure learning, however in this work we follow the latter and separate train/test data. (4) evaluation: for unlabeled F_1 , almost all works ignore punctuation (even approaches that use punctuation during training typically ignore them during evaluation), but there is some variance in discarding trivial spans (width-one and sentence-level spans) and using corpus-

⁶However, we used F_1 against validation trees on PTB to select some hyperparameters (e.g. grammar size), as is sometimes done in grammar induction. Hence our PTB results are arguably not fully unsupervised in the strictest sense of the term. The hyperparameters of the PRPN/ON baselines are also tuned using validation F_1 for fair comparison.

level versus sentence-level F_1 .⁷ In this paper we discard trivial spans and evaluate on sentence-level F_1 per recent work (Shen et al., 2018b, 2019). Notably the sentence-level F_1 differs from the corpus-level F_1 calculated by `evalb` from the previous chapter.

Given the above, we mainly compare our approach against two recent, strong baselines with open source code: Parsing Predict Reading Network (PRPN)⁸ (Shen et al., 2018b) and Ordered Neurons (ON)⁹ (Shen et al., 2019). These approaches train a neural language model with gated attention-like mechanisms to induce binary trees, and achieve strong unsupervised parsing performance even when trained on corpora where punctuation is removed. Since the original results were on both language modeling and grammar induction, their hyperparameters were presumably tuned to do well on both and thus may not be optimal for just unsupervised parsing. We therefore tune the hyperparameters of these baselines for unsupervised parsing only (i.e. on validation F_1).

CODE Our code is available at <https://github.com/harvardnlp/compound-pcfg>.

6.4.2 RESULTS

Table 6.1 shows the unlabeled F_1 scores for our models and various baselines. All models soundly outperform right branching baselines, and we find that the neural/compound PCFGs are strong models for grammar induction. In particular the compound PCFG outperforms other models by an appreciable margin on both English and Chinese. We again note that we were unable to induce meaningful grammars through a traditional

⁷Corpus-level F_1 calculates precision/recall at the corpus level to obtain F_1 , while sentence-level F_1 calculates F_1 for each sentence and averages across the corpus.

⁸<https://github.com/yikangshen/PRPN>

⁹<https://github.com/yikangshen/Ordered-Neurons>

Model	PTB		CTB	
	Mean	Max	Mean	Max
PRPN (Shen et al., 2018b)	37.4	38.1	—	—
ON (Shen et al., 2019)	47.7	49.4	—	—
URNNG [†] (Kim et al., 2019b)	—	45.4	—	—
DIORA [†] (Drozдов et al., 2019)	—	58.9	—	—
Left Branching	8.7		9.7	
Right Branching	39.5		20.0	
Random Trees	19.2	19.5	15.7	16.0
PRPN (tuned)	47.3	47.9	30.4	31.5
ON (tuned)	48.1	50.0	25.4	25.7
Scalar PCFG	< 35.0		< 15.0	
Neural PCFG	50.8	52.6	25.7	29.5
Compound PCFG	55.2	60.1	36.0	39.8
Oracle Trees	84.3		81.1	

Table 6.1: Unlabeled sentence-level F_1 scores on English (PTB) and Chinese (CTB) test sets. Top shows results from previous work while the rest of the results are from this paper. Mean/Max scores are obtained from 4 runs of each model with different random seeds. Oracle is the maximum score obtainable with binarized trees, since we compare against the non-binarized gold trees per convention. Results with [†] are trained on a version of PTB with punctuation, and hence not strictly comparable to the present work. For URNNG/DIORA, we take the parsed test set provided by the authors from their best runs and evaluate F_1 with our evaluation setup, which ignores punctuation.

PCFG with the scalar parameterization despite a thorough hyperparameter search.¹⁰

Table 6.2 analyzes the learned tree structures. We compare similarity as measured by F_1 against gold, left, right, and “self” trees (top), where self F_1 score is calculated by averaging over all 6 pairs obtained from 4 different runs. We find that PRPN is particularly consistent across multiple runs. We also observe that different models are better at identifying different constituent labels, as measured by label recall (Table 6.2, bottom). While left as future work, this naturally suggests an ensemble ap-

¹⁰Training perplexity was much higher than in the neural case, indicating significant optimization issues.

	PRPN	ON	Neural PCFG	Compound PCFG
Gold	47.3	48.1	50.8	55.2
Left	1.5	14.1	11.8	13.0
Right	39.9	31.0	27.7	28.4
Self	82.3	71.3	65.2	66.8
SBAR	50.0%	51.2%	52.5%	56.1%
NP	59.2%	64.5%	71.2%	74.7%
VP	46.7%	41.0%	33.8%	41.7%
PP	57.2%	54.4%	58.8%	68.8%
ADJP	44.3%	38.1%	32.5%	40.4%
ADVP	32.8%	31.6%	45.5%	52.5%

Table 6.2: (Top) Mean F_1 similarity against Gold, Left, Right, and Self trees on the English PTB. Self F_1 score is calculated by averaging over all 6 pairs obtained from 4 different runs. (Bottom) Fraction of ground truth constituents that were predicted as a constituent by the models broken down by label (i.e. label recall).

proach wherein the empirical probabilities of constituents (obtained by averaging the predicted binary constituent labels from the different models) are used either to supervise another model or directly as potentials in a CRF constituency parser. Finally, all models seemed to have some difficulty in identifying SBAR/VP constituents which typically span more words than NP constituents, indicating opportunities for further improvements in unsupervised parsing.

6.4.3 ANALYSIS

NONTERMINAL ALIGNMENT Since we induce a full set of nonterminals in our grammar, we can analyze the learned nonterminals to see if they can be aligned with linguistic constituent labels. Table 6.3 shows the alignment between induced and gold labels in the neural PCFG, where for each nonterminal we show the empirical probability that a predicted constituent of this type will correspond to a particular lin-

Label	S	SBAR	NP	VP	PP	ADJP	ADVP	OTHER	Freq.	Acc.
NT-01	0.0%	0.0%	81.8%	1.1%	0.0%	5.9%	0.0%	11.2%	2.9%	13.8%
NT-02	2.2%	0.9%	90.8%	1.7%	0.9%	0.0%	1.3%	2.2%	1.1%	44.0%
NT-03	1.0%	0.0%	2.3%	96.8%	0.0%	0.0%	0.0%	0.0%	1.8%	37.1%
NT-04	0.3%	2.2%	0.5%	2.0%	93.9%	0.2%	0.6%	0.3%	11.0%	64.9%
NT-05	0.2%	0.0%	36.4%	56.9%	0.0%	0.0%	0.2%	6.2%	3.1%	57.1%
NT-06	0.0%	0.0%	99.1%	0.0%	0.1%	0.0%	0.2%	0.6%	5.2%	89.0%
NT-07	0.0%	0.0%	99.7%	0.0%	0.3%	0.0%	0.0%	0.0%	1.3%	59.3%
NT-08	0.5%	2.2%	23.3%	35.6%	11.3%	23.6%	1.7%	1.7%	2.0%	44.3%
NT-09	6.3%	5.6%	40.2%	4.3%	32.6%	1.2%	7.0%	2.8%	2.6%	52.1%
NT-10	0.1%	0.1%	1.4%	58.8%	38.6%	0.0%	0.8%	0.1%	3.0%	50.5%
NT-11	0.9%	0.0%	96.5%	0.9%	0.9%	0.0%	0.0%	0.9%	1.1%	42.9%
NT-12	0.5%	0.2%	94.4%	2.4%	0.2%	0.1%	0.2%	2.0%	8.9%	74.9%
NT-13	1.6%	0.1%	0.2%	97.7%	0.2%	0.1%	0.1%	0.1%	6.2%	46.0%
NT-14	0.0%	0.0%	0.0%	98.6%	0.0%	0.0%	0.0%	1.4%	0.9%	54.1%
NT-15	0.0%	0.0%	99.7%	0.0%	0.3%	0.0%	0.0%	0.0%	2.0%	76.9%
NT-16	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.3%	29.9%
NT-17	96.4%	2.9%	0.0%	0.7%	0.0%	0.0%	0.0%	0.0%	1.2%	24.4%
NT-18	0.3%	0.0%	88.7%	2.8%	0.3%	0.0%	0.0%	7.9%	3.0%	28.3%
NT-19	3.9%	1.0%	86.6%	2.4%	2.6%	0.4%	1.3%	1.8%	4.5%	53.4%
NT-20	0.0%	0.0%	99.0%	0.0%	0.0%	0.3%	0.2%	0.5%	7.4%	17.5%
NT-21	94.4%	1.7%	2.0%	1.4%	0.3%	0.1%	0.0%	0.1%	6.2%	34.7%
NT-22	0.1%	0.0%	98.4%	1.1%	0.1%	0.0%	0.2%	0.2%	3.5%	77.6%
NT-23	0.4%	0.9%	14.0%	53.1%	8.2%	18.5%	4.3%	0.7%	2.4%	49.1%
NT-24	0.0%	0.2%	1.5%	98.3%	0.0%	0.0%	0.0%	0.0%	2.3%	47.3%
NT-25	0.3%	0.0%	1.4%	98.3%	0.0%	0.0%	0.0%	0.0%	2.2%	34.6%
NT-26	0.4%	60.7%	18.4%	3.0%	15.4%	0.4%	0.4%	1.3%	2.1%	23.4%
NT-27	0.0%	0.0%	48.7%	0.5%	0.7%	13.1%	3.2%	33.8%	2.0%	59.7%
NT-28	88.2%	0.3%	3.8%	0.9%	0.1%	0.0%	0.0%	6.9%	6.7%	76.5%
NT-29	0.0%	1.7%	95.8%	1.0%	0.7%	0.0%	0.0%	0.7%	1.0%	62.8%
NT-30	1.6%	94.5%	0.6%	1.2%	1.2%	0.0%	0.4%	0.4%	2.1%	49.4%
Gold	15.0%	4.8%	38.5%	21.7%	14.6%	1.7%	0.8%	2.9%		

Table 6.3: Analysis of label alignment for nonterminals in the neural PCFG. Label alignment is the proportion of correctly-predicted constituents that correspond to a particular gold label. We also show the predicted constituent frequency and accuracy (i.e. precision) on the rightmost two columns. Bottom line shows the empirical frequency of the constituent labels in gold trees.

guistic constituent in the test set, conditioned on its being a correct constituent (for reference we also show the precision). Table 6.4 has a similar alignment table for the compound PCFG. We observe that some of the induced nonterminals clearly align to linguistic nonterminals. We also observed the preterminals to align to part-of-speech tags (not shown).¹¹

¹¹As a POS induction system, the many-to-one performance of the compound PCFG using the preterminals is 68.0. A similarly-parameterized compound HMM with 60 hidden states (an HMM is a particularly type of PCFG) obtains 63.2. This is still quite a bit lower than the

Label	S	SBAR	NP	VP	PP	ADJP	ADVP	OTHER	Freq.	Acc.
NT-01	0.0%	0.0%	0.0%	99.2%	0.0%	0.0%	0.0%	0.8%	2.6%	41.1%
NT-02	0.0%	0.3%	0.3%	99.2%	0.0%	0.0%	0.0%	0.3%	5.3%	15.4%
NT-03	88.2%	0.3%	3.6%	1.0%	0.1%	0.0%	0.0%	6.9%	7.2%	71.4%
NT-04	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.5%	2.4%
NT-05	0.0%	0.0%	0.0%	96.6%	0.0%	0.0%	0.0%	3.4%	5.0%	1.2%
NT-06	0.0%	0.4%	0.4%	98.8%	0.0%	0.0%	0.0%	0.4%	1.2%	43.7%
NT-07	0.2%	0.0%	95.3%	0.9%	0.0%	1.6%	0.1%	1.9%	2.8%	60.6%
NT-08	1.0%	0.4%	95.3%	2.3%	0.4%	0.2%	0.3%	0.2%	9.4%	63.0%
NT-09	0.6%	0.0%	87.4%	1.9%	0.0%	0.0%	0.0%	10.1%	1.0%	33.8%
NT-10	78.3%	17.9%	3.0%	0.5%	0.0%	0.0%	0.0%	0.3%	1.9%	42.0%
NT-11	0.3%	0.0%	99.0%	0.3%	0.0%	0.3%	0.0%	0.0%	0.9%	70.3%
NT-12	0.0%	8.8%	76.5%	2.9%	5.9%	0.0%	0.0%	5.9%	2.0%	3.6%
NT-13	0.5%	2.0%	1.0%	96.6%	0.0%	0.0%	0.0%	0.0%	1.7%	50.7%
NT-14	0.0%	0.0%	99.1%	0.0%	0.0%	0.6%	0.0%	0.4%	7.7%	14.8%
NT-15	2.9%	0.5%	0.4%	95.5%	0.4%	0.0%	0.0%	0.2%	4.4%	45.2%
NT-16	0.4%	0.4%	17.9%	5.6%	64.1%	0.4%	6.8%	4.4%	1.4%	38.1%
NT-17	0.1%	0.0%	98.2%	0.5%	0.1%	0.1%	0.1%	0.9%	9.6%	85.4%
NT-18	0.1%	0.0%	95.7%	1.6%	0.0%	0.1%	0.2%	2.3%	4.7%	56.2%
NT-19	0.0%	0.0%	98.9%	0.0%	0.4%	0.0%	0.0%	0.7%	1.3%	72.6%
NT-20	2.0%	22.7%	3.0%	4.8%	63.9%	0.6%	2.3%	0.6%	6.8%	59.0%
NT-21	0.0%	0.0%	14.3%	42.9%	0.0%	0.0%	42.9%	0.0%	2.2%	0.7%
NT-22	1.4%	0.0%	11.0%	86.3%	0.0%	0.0%	0.0%	1.4%	1.0%	15.2%
NT-23	0.1%	0.0%	58.3%	0.8%	0.4%	5.0%	1.7%	33.7%	2.8%	62.7%
NT-24	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.6%	70.2%
NT-25	2.2%	0.0%	76.1%	4.3%	0.0%	2.2%	0.0%	15.2%	0.4%	23.5%
NT-26	0.0%	0.0%	2.3%	94.2%	3.5%	0.0%	0.0%	0.0%	0.8%	24.0%
NT-27	96.6%	0.2%	1.5%	1.1%	0.3%	0.2%	0.0%	0.2%	4.3%	32.2%
NT-28	1.2%	3.7%	1.5%	5.8%	85.7%	0.9%	0.9%	0.3%	7.6%	64.9%
NT-29	3.0%	82.0%	1.5%	13.5%	0.0%	0.0%	0.0%	0.0%	0.6%	45.4%
NT-30	0.0%	0.0%	1.0%	60.2%	19.4%	1.9%	4.9%	12.6%	2.1%	10.4%
Gold	15.0%	4.8%	38.5%	21.7%	14.6%	1.7%	0.8%	2.9%		

Table 6.4: Analysis of label alignment for nonterminals in the compound PCFG. Label alignment is the proportion of correctly-predicted constituents that correspond to a particular gold label. We also show the predicted constituent frequency and accuracy (i.e. precision) on the rightmost two columns. Bottom line shows the empirical frequency of the constituent labels in gold trees.

CONTINUOUS LATENT SPACE We next investigate what is being encoded in the continuous latent space in the compound PCFG. Table 6.5 shows nearest neighbors of some sentences using the mean of the variational posterior as the continuous representation of each sentence. We qualitatively observe that the latent space seems to capture topical information. We are also interested in the variation in the leaves

state-of-the-art (He et al., 2018; Stratos, 2019), though comparison is confounded by various factors such as preprocessing (e.g. we drop punctuation). A neural PCFG/HMM obtains 68.2 and 63.4 respectively.

<p>he retired as senior vice president finance and administration and chief financial officer of the company oct. N kenneth j. (unk) who was named president of this thrift holding company in august resigned citing personal reasons the former president and chief executive eric w. (unk) resigned in june (unk) 's president and chief executive officer john (unk) said the loss stems from several factors mr. (unk) is executive vice president and chief financial officer of (unk) and will continue in those roles charles j. lawson jr. N who had been acting chief executive since june N will continue as chairman</p>
<p>(unk) corp. received an N million army contract for helicopter engines boeing co. received a N million air force contract for developing cable systems for the (unk) missile general dynamics corp. received a N million air force contract for (unk) training sets grumman corp. received an N million navy contract to upgrade aircraft electronics thomson missile products with about half british aerospace 's annual revenue include the (unk) (unk) missile family already british aerospace and french (unk) (unk) (unk) on a british missile contract and on an air-traffic control radar system</p>
<p>meanwhile during the the s&p trading halt s&p futures sell orders began (unk) up while stocks in new york kept falling sharply but the (unk) of s&p futures sell orders weighed on the market and the link with stocks began to fray again on friday some market makers were selling again traders said futures traders say the s&p was (unk) that the dow could fall as much as N points meanwhile two initial public offerings (unk) the (unk) market in their (unk) day of national over-the-counter trading friday traders said most of their major institutional investors on the other hand sat tight</p>

Table 6.5: For each query sentence (bold), we show the 5 nearest neighbors based on cosine similarity, where we take the representation for each sentence to be the mean of the variational posterior.

due to \mathbf{z} when the variation due to the tree structure is held constant. To investigate this, we use the parsed dataset to obtain pairs of the form $(\mu_\phi(x^{(n)}), \mathbf{t}_j^{(n)})$, where $\mathbf{t}_j^{(n)}$ is the j -th subtree of the (approximate) MAP tree $\mathbf{t}^{(n)}$ for the n -th sentence. Therefore each mean vector $\mu_\phi(x^{(n)})$ is associated with $|x^{(n)}| - 1$ subtrees, where $|x^{(n)}|$ is the sentence length. Our definition of subtree here ignores terminals, and thus each subtree is associated with many mean vectors. For a frequently occurring subtree, we perform PCA on the set of mean vectors that are associated with the subtree to obtain the top principal component. We then show the constituents that had the 5 most positive/negative values for this top principal component in Table 6.6. For example, a particularly common subtree—associated with 180 unique constituents—is given by

$$\begin{aligned}
&(\text{NT-04 } (T-13 \ w_1) \ (\text{NT-12 } (\text{NT-20 } (\text{NT-20 } (\text{NT-07 } (T-05 \ w_2) \\
&\quad (T-45 \ w_3)) \ (T-35 \ w_4)) \ (T-40 \ w_5)) \ (T-22 \ w_6))).
\end{aligned}$$

The top 5 constituents with the most negative/positive values are shown at the top of Table 6.6. We find that the leaves $[w_1, \dots, w_6]$, which form a 6-word constituent, vary

in a regular manner as \mathbf{z} is varied. We also observe that root of this subtree (NT-04) aligns to prepositional phrases (PP) in Table 6.4, and the leaves in Table 6.6 (top left) are indeed mostly prepositional phrases. However, the model fails to identify ((T-40 w_5) (T-22 w_6)) as a constituent in this case. It is possible that the model is utilizing the subtrees to capture broad template-like structures and then using \mathbf{z} to fill them in, similar to recent works that also train models to separate “what to say” from “how to say it” (Wiseman et al., 2018; Peng et al., 2019; Chen et al., 2019a,b).

6.4.4 RECURRENT NEURAL NETWORK GRAMMARS ON INDUCED TREES

While the compound PCFG has fewer independence assumptions than the neural PCFG, it is still a more constrained model of language than recurrent neural network grammars from the previous chapter (or standard neural language models for that matter), and thus not competitive in terms of perplexity: the compound PCFG obtains a perplexity of 196.3 while an LSTM language model (LM) obtains 86.2 (Table 6.7).¹² In contrast, both PRPN and ON perform as well as an LSTM LM while maintaining good unsupervised parsing performance.

We thus experiment to see if it is possible to use the induced trees to supervise a more flexible generative model that can make use of tree structures, in particular recurrent neural network grammars (RNNG) from chapter 5. We take the best run from each model and parse the training set,¹³ and use the induced trees to supervise

¹²It was possible to almost match the perplexity of an NLM by additionally conditioning the terminal probabilities on previous history,

$$\pi_{\mathbf{z}, T \rightarrow w_t} \propto \exp(\mathbf{u}_w^\top \text{MLP}_2([\mathbf{w}_T; \mathbf{z}; \mathbf{h}_t; \theta]) + b_w),$$

where \mathbf{h}_t is the hidden state from an LSTM over $x_{<t}$. However the unsupervised parsing performance was far worse ($\approx 25 F_1$ on the PTB).

¹³The train/test F_1 was similar for all models.

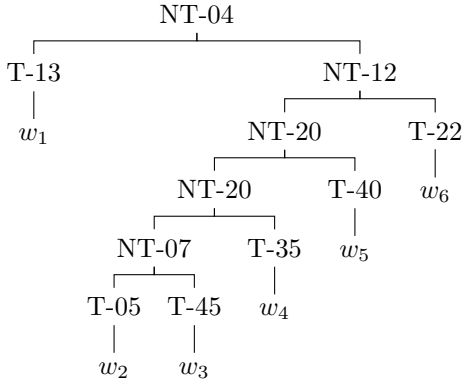
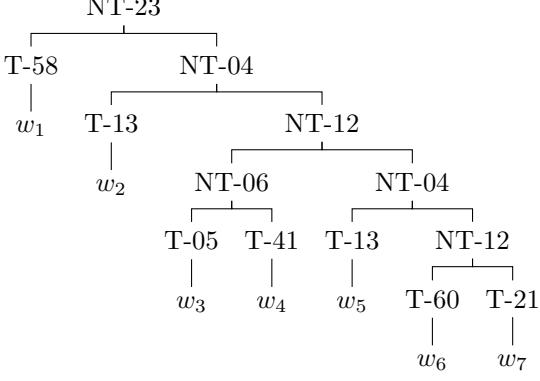
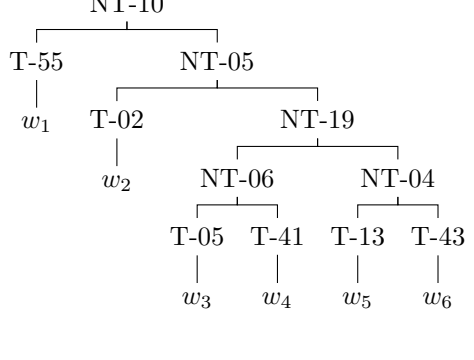
	<p>PC -</p> <ul style="list-style-type: none"> of the company 's capital structure in the company 's divestiture program by the company 's new board in the company 's core businesses on the company 's strategic plan <p>PC +</p> <ul style="list-style-type: none"> above the treasury 's N-year note above the treasury 's seven-year note above the treasury 's comparable note above the treasury 's five-year note measured the earth 's ozone layer
	<p>PC -</p> <ul style="list-style-type: none"> purchased through the exercise of stock options circulated by a handful of major brokers higher as a percentage of total loans common with a lot of large companies surprised by the storm of sell orders <p>PC +</p> <ul style="list-style-type: none"> brought to the u.s. against her will laid for the arrest of opposition activists uncertain about the magnitude of structural damage held after the assassination of his mother hurt as a result of the violations
	<p>PC -</p> <ul style="list-style-type: none"> to terminate their contract with warner to support a coup in panama to suit the bureaucrats in brussels to thwart his bid for amr to prevent the pound from rising <p>PC +</p> <ul style="list-style-type: none"> to change our strategy of investing to offset the growth of minimills to be a lot of art to change our way of life to increase the impact of advertising

Table 6.6: For each subtree, we perform PCA on the variational posterior mean vectors that are associated with that particular subtree and take the top principal component. We then list the top 5 constituents that had the lowest (**PC -**) and highest (**PC +**) principal component values.

	PPL	Syntactic Eval.	F_1
LSTM LM	86.2	60.9%	—
PRPN	87.1	62.2%	47.9
RNNG	95.3	60.1%	47.8
RNNG \rightarrow URNNG	90.1	61.8%	51.6
ON	87.2	61.6%	50.0
RNNG	95.2	61.7%	50.6
RNNG \rightarrow URNNG	89.9	61.9%	55.1
Neural PCFG	252.6	49.2%	52.6
RNNG	95.8	68.1%	51.4
RNNG \rightarrow URNNG	86.0	69.1%	58.7
Compound PCFG	196.3	50.7%	60.1
RNNG	89.8	70.0%	58.1
RNNG \rightarrow URNNG	83.7	76.1%	66.9
Oracle Trees			
RNNG	80.6	70.4%	71.9
RNNG \rightarrow URNNG	78.3	76.1%	72.8

Table 6.7: Results from training RNNGs on induced trees from various models on the PTB. RNNG \rightarrow URNNG indicates fine-tuning with the URNNG objective. We show perplexity (PPL), grammaticality judgment performance (Syntactic Eval.), and unlabeled F_1 . PPL/ F_1 are calculated on the PTB test set and Syntactic Eval. is from [Marvin & Linzen \(2018\)](#)’s dataset. Results on top do not make any use of annotated trees, while the bottom two results are trained on binarized gold trees. All the RNN-based models above (i.e. LSTM/PRPN/ON/RNNG/URNNG) have roughly the same model capacity. Note that the results here are not comparable to Table 5.1 and Table 5.5 from chapter 5 due to differences in vocabulary and preprocessing.

an RNNG for each model, using the same parameterization and training setup from the previous chapter. We test the induced RNNGs on syntactic evaluation, again utilizing the framework and dataset from [Marvin & Linzen \(2018\)](#). As in chapter 5 we also train a version of the model where we fine-tune the RNNG with the unsupervised RNNG objective (RNNG \rightarrow URNNG). See chapter 5 for the exact details.

The results are shown in Table 6.7. For perplexity, RNNGs trained on induced trees (Induced RNNG in Table 6.7) are unable to improve upon an LSTM LM, in contrast to the supervised RNNG which does outperform the LSTM language model

(Table 6.7, bottom).¹⁴ For grammaticality judgment however, the RNNG trained with compound PCFG trees outperforms the LSTM LM despite obtaining worse perplexity,¹⁵ and performs on par with the RNNG trained on binarized gold trees. Fine-tuning with the URNNG results in improvements in perplexity and grammaticality judgment across the board (RNNG \rightarrow URNNG in Table 6.7). We also obtain large improvements on unsupervised parsing as measured by F_1 , with the fine-tuned URNNGs outperforming the respective original models.¹⁶ This is potentially due to an ensembling effect between the original model and the URNNG’s structured inference network, which is parameterized as a neural CRF constituency parser (Durrett & Klein, 2015; Liu et al., 2018). While left as future work, it is possible to use the compound PCFG itself as an inference network. Finally, as noted in the previous chapter, an unsupervised RNNG trained from scratch fails to outperform a right-branching baseline on this version of PTB where punctuation is removed (not shown).

6.5 DISCUSSION

6.5.1 LIMITATIONS

We report on some negative results as well as important limitations of our approach in this section. While distributed representations promote parameter sharing, we were unable to obtain improvements through more factorized parameterizations that promote even greater parameter sharing. In particular, for rules of the type $A \rightarrow BC$,

¹⁴We again note that these results are not comparable to the perplexity results from Table 5.1 since the vocabulary size and preprocessing is different.

¹⁵Kuncoro et al. (2018, 2019) also observe that models that achieve lower perplexity do not necessarily perform better on syntactic evaluation tasks.

¹⁶Li et al. (2019c) similarly obtain improvements by refining a model trained on induced trees on classification tasks.

we tried having the output embeddings be a function of the input embeddings (e.g. $\mathbf{u}_{BC} = \text{MLP}([\mathbf{w}_B; \mathbf{w}_C; \theta])$), but obtained worse results. For rules of the type $T \rightarrow w$, we tried using a character-level CNN (dos Santos & Zadrozny, 2014; Kim et al., 2016) to obtain the output word embeddings \mathbf{u}_w , but found the performance to be similar to the word-level case.¹⁷ We were also unable to obtain improvements through normalizing flows (Rezende & Mohamed, 2015b; Kingma et al., 2016). However, given that we did not exhaustively explore the full space of possible parameterizations, the above modifications could eventually lead to improvements with the right setup.

Relatedly, the models were quite sensitive to parameterization (e.g. it was important to use residual layers for the MLPs), grammar size, and optimization method. We also noticed some variance in results across random seeds, as shown in Table 6.2. Finally, despite vectorized GPU implementations, training was significantly more expensive (both in terms of time and memory) than NLM-based grammar induction systems due to the $O(|\mathcal{R}||x|^3)$ dynamic program, which makes our approach potentially difficult to scale.

6.5.2 RICHER GRAMMARS FOR MODELING NATURAL LANGUAGE

We initially motivated the compound PCFG as an approximation to richer PCFG formalisms such as higher-order (Klein & Manning, 2003) and lexicalized (Collins, 1997) PCFGs. While neural parameterization and inference made it computationally straightforward to approximately model richer interactions in a “soft” way with a sentence-level latent vector, in some sense these vector-based representations are less

¹⁷It is also possible to take advantage of pretrained word embeddings by using them to initialize output word embeddings or directly working with continuous emission distributions (Lin et al., 2015; He et al., 2018)

interpretable compared to the discrete nonterminal symbols.¹⁸ It is well known that some natural language phenomena cannot be modeled with context-free grammars (Shieber, 1985). A possible future direction would therefore involve directly modeling richer grammars that still admit polynomial-time inference algorithms. For example, it would be interesting to explore neural parameterizations of mildly context-sensitive formalisms such as tree adjoining grammars (Joshi et al., 1975), combinatory categorical grammars (Steedman, 1987), or head grammars (Pollard, 1984).¹⁹

6.6 RELATED WORK

Grammar induction and unsupervised parsing have a long and rich history in natural language processing. Early work on grammar induction with pure unsupervised learning was mostly negative (Lari & Young, 1990; Carroll & Charniak, 1992; Charniak, 1993), though Pereira & Schabes (1992) reported some success on partially bracketed data. Clark (2001) and Klein & Manning (2002) were some of the first successful statistical approaches to unsupervised parsing. In particular, the constituent-context model (CCM) of Klein & Manning (2002), which explicitly models both constituents and distituent, was the basis for much subsequent work (Klein & Manning, 2004; Huang et al., 2012; Golland et al., 2012). Other works have explored imposing inductive biases through Bayesian priors (Johnson et al., 2007; Liang et al., 2007; Wang & Blunsom, 2013), modified objectives (Smith & Eisner, 2004), and additional constraints on recursion depth (Noji et al., 2016; Jin et al., 2018b,a).

While the framework of specifying the structure of a grammar and learning the pa-

¹⁸An interesting theoretical question would be to investigate the formal class of languages generated by compound PCFGs.

¹⁹These formalisms are all weakly equivalent in that they generate the same languages (sets of strings) (Vijay-Shanker & Weir, 1994).

rameters is common, other methods exist. Bod (2006) consider a nonparametric-style approach to unsupervised parsing by using random subsets of training subtrees to parse new sentences. Seginer (2007) utilize an incremental algorithm to unsupervised parsing which makes local decisions to create constituents based on a complex set of heuristics. Ponvert et al. (2011) induce parse trees through cascaded applications of finite state models. More recently, neural network-based approaches to unsupervised parsing have shown promising results on inducing parse trees directly from words. Shen et al. (2018b, 2019) learn tree structures through soft gating layers within neural language models, while Drozdov et al. (2019) combine recursive autoencoders with the inside-outside algorithm. Shi et al. (2019) utilize image captions to identify and ground constituents.

The compound PCFG outlined in this chapter is also related to latent variable PCFGs (Matsuzaki et al., 2005; Petrov et al., 2006; Cohen et al., 2012), which extend PCFGs to the latent variable setting by splitting nonterminal symbols into latent sub-symbols. In particular, latent vector grammars (Zhao et al., 2018) and compositional vector grammars (Socher et al., 2013a) also employ continuous vectors within their grammars. However these approaches have been employed for learning supervised parsers on annotated treebanks, unlike the unsupervised setting of the current work.

6.7 CONCLUSION

This chapter explores grammar induction with compound PCFGs, which modulate rule probabilities with per-sentence continuous latent vectors. The latent vector induces marginal dependencies beyond the traditional first-order context-free assumptions within a tree-based generative process, leading to improved performance. The

collapsed amortized variational inference approach is general and can be used for generative models which admit tractable inference through partial conditioning. Learning deep generative models which exhibit such conditional Markov properties is an interesting direction for future work.

7

Conclusion

This thesis has explored deep latent variable models of natural language. We have shown that these models can be used to model a wide range of language phenomena, from attention in neural machine translation to parse trees and even full grammars. Amortized variational inference, in which a global inference network is trained to output the parameters of an approximate variational posterior distribution, was a key tool in enabling efficient learning of deep latent variable models. However, a straightforward application of amortized variational inference often proved insufficient for

many language applications of interest, and in each chapter we studied an extension of the standard approach : (1) semi-amortized variational inference in chapter 3, (2) continuous relaxations of discrete distributions in chapter 4, (3) posterior regularization with a structured inference network in chapter 5, and (4) collapsed amortized variational inference in chapter 6. From an applications standpoint, we showed how these models can be applied to a range of core NLP tasks including language modeling, machine translation, and unsupervised parsing.

Deep latent variable models, which combine the composability and interpretability of latent variable models with the flexible modeling capabilities of deep networks, are an exciting area of research. It is nonetheless worth discussing again *why* we would want to use latent variable models in the first place, especially given that from a pure performance standpoint, models that do not formally make use of latent variables are incredibly effective. Even from the perspective of learning interesting/meaningful structures (e.g. topics, alignments, phrases), one may argue that these structures can be captured implicitly within the hidden layers of a deep network through end-to-end learning. Indeed, the recent, astonishing success of deep, *non*-latent variable models pretrained with self-supervised objectives (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018) suggests that deep networks can capture a significant amount of linguistic knowledge without the explicit use of latent variables.

We conclude this thesis by offering a spirited defense of latent variable models in contemporary NLP. For one, latent variable models *can* surpass the performance of deterministic deep networks if properly optimized. For example, deterministic (soft) attention (Bahdanau et al., 2015) was generally thought to outperform latent variable (hard) attention (Xu et al., 2015), until recent work showed that when properly optimized (albeit with a more expensive training procedure), latent variable attention can

outperform deterministic attention (Deng et al., 2018; Shankar et al., 2018; Wu et al., 2018). Framing certain problems as essentially approximating a latent variable objective often yields valuable insights and new avenues for further work. Notably, the interpretation of dropout (Hinton et al., 2012; Srivastava et al., 2014) as optimizing a latent variable objective has led to rich extensions and improvements (Kingma et al., 2015; Gal & Ghahramani, 2016a,b; Ma et al., 2017; Melis et al., 2018a). Finally, while there is much evidence that non-latent variable models pretrained with self-supervised objectives are able to implicitly capture a significant amount of linguistic knowledge within their hidden layers (Liu et al., 2019; Hewitt & Manning, 2019; Tenney et al., 2019), it would be ideal to have more direct, *explicit* access to such structures through latent variables, especially from the perspective of interpretability, transparency, controllability, and transfer learning.

More generally, latent variable modeling gives us a modular, probabilistic framework with which to explicitly inject both *inductive bias* and domain-specific *constraints* into models. Inductive bias, or the inherent “preferences” of a model (or learning algorithm), is crucial for learning and generalization. It can help mitigate against model misspecification, allow for data-efficient learning, and through a carefully crafted generative model, enable interesting structures to emerge. Furthermore, if we have constraints on the representations learned by a model, such as that they represent a valid parse tree, or be interpretable, or allow for controlling the model’s predictions, we can enforce these constraints in a principled way through latent variables. The intersection of latent variable models and deep learning remains an exciting avenue for much future work.

References

- Aharoni, R. & Goldberg, Y. (2017). Towards String-to-Tree Neural Machine Translation. In *Proceedings of ACL*.
- Aho, A. (1968). Indexed Grammars—An Extension of Context-Free Grammars. *Journal of the ACM*, 15(4), 647–671.
- Alvarez-Melis, D. & Jaakkola, T. S. (2017a). A Causal Framework for Explaining the Predictions of Black-Box Sequence-to-Sequence Models. In *Proceedings of EMNLP*.
- Alvarez-Melis, D. & Jaakkola, T. S. (2017b). Tree-structured Decoding with Doubly-Recurrent Neural Networks. In *Proceedings of ICLR*.
- Ammar, W., Dyer, C., & Smith, N. A. (2014). Conditional Random Field Autoencoders for Unsupervised Structured Prediction. In *Proceedings of NIPS*.
- Amos, B. & Kolter, J. Z. (2017). OptNet: Differentiable Optimization as a Layer in Neural Networks. In *Proceedings of ICML*.
- Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M., Pfau, D., Schaul, T., & de Freitas, N. (2016). Learning to Learn by Gradient Descent by Gradient Descent. In *Proceedings of NIPS*.
- Arora, S., Cohen, N., & Hazan, E. (2018). On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization. In *Proceedings of ICML*.
- Ba, J., Mnih, V., & Kavukcuoglu, K. (2015a). Multiple Object Recognition with Visual Attention. In *Proceedings of ICLR*.
- Ba, J., Salakhutdinov, R. R., Grosse, R. B., & Frey, B. J. (2015b). Learning Wake-Sleep Recurrent Attention Models. In *Proceedings of NIPS*.
- Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer Normalization. *arXiv:1607.06450*.
- Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of ICLR*.

- Bahuleyan, H., Mou, L., Vechtomova, O., & Poupart, P. (2017). Variational Attention for Sequence-to-Sequence Models. *arXiv:1712.08207*.
- Baker, J. K. (1979). Trainable Grammars for Speech Recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*.
- Belanger, D., Yang, B., & McCallum, A. (2017). End-to-end Learning for Structured Prediction Energy Networks. In *Proceedings of ICML*.
- Bengio, Y., Ducharme, R., & Vincent, P. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3, 1137–1155.
- Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum Learning. In *Proceedings of ICML*.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
- Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, 112(518), 859–877.
- Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan), 993–1022.
- Bod, R. (2006). An All-Subtrees Approach to Unsupervised Parsing. In *Proceedings of ACL*.
- Bojar, O., Buck, C., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huck, M., Yepes, A. J., Koehn, P., & Kreutzer, J. (2017). Proceedings of the second conference on machine translation. In *Proceedings of the Second Conference on Machine Translation: Association for Computational Linguistics*.
- Bornschein, J., Mnih, A., Zoran, D., & Rezende, D. J. (2017). Variational Memory Addressing in Generative Models. In *Proceedings of NIPS*.
- Bowman, S. R., Vilnis, L., Vinyal, O., Dai, A. M., Jozefowicz, R., & Bengio, S. (2016). Generating Sentences from a Continuous Space. In *Proceedings of CoNLL*.
- Bradbury, J. & Socher, R. (2017). Towards Neural Machine Translation with Latent Tree Attention. In *Proceedings of the 2nd Workshop on Structured Prediction for Natural Language Processing*.
- Brakel, P., Stroobandt, D., & Schrauwen, B. (2013). Training Energy-Based Models for Time-Series Imputation. *Journal of Machine Learning Research*, 14, 2771–2797.

- Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., & Lai, J. C. (1992). Class-based N-gram Models of Natural Language. *Computational Linguistics*, 18(4), 467–479.
- Brown, P. F., Pietra, S. A. D., Pietra, V. J. D., & Mercer, R. L. (1993). The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational linguistics*, 19(2), 263–311.
- Burda, Y., Grosse, R., & Salakhutdinov, R. (2015). Importance Weighted Autoencoders. In *Proceedings of ICLR*.
- Buys, J. & Blunsom, P. (2015). Generative Incremental Dependency Parsing with Neural Networks. In *Proceedings of ACL*.
- Buys, J. & Blunsom, P. (2018). Neural Syntactic Generative Models with Exact Marginalization. In *Proceedings of NAACL*.
- Cai, J., Jiang, Y., & Tu, K. (2017). CRF Autoencoder for Unsupervised Dependency Parsing. In *Proceedings of EMNLP*.
- Carroll, G. & Charniak, E. (1992). Two Experiments on Learning Probabilistic Dependency Grammars from Corpora. In *AAAI Workshop on Statistically-Based NLP Techniques*.
- Cettolo, M., Niehues, J., Stuker, S., Bentivogli, L., & Federico, M. (2014). Report on the 11th IWSLT evaluation campaign. In *Proceedings of IWSLT*.
- Charniak, E. (1993). *Statistical Language Learning*. MIT Press.
- Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., & Robinson, T. (2013). One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. *arXiv:1312.3005*.
- Chen, D., Fisch, A., Weston, J., & Bordes, A. (2017a). Reading Wikipedia to answer open-domain questions. In *Proceedings of ACL*.
- Chen, D. & Manning, C. D. (2014). A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of EMNLP*.
- Chen, M., Tang, Q., Wiseman, S., & Gimpel, K. (2019a). Controllable Paraphrase Generation with a Syntactic Exemplar. In *Proceedings of ACL*.
- Chen, M., Tang, Q., Wiseman, S., & Gimpel, K. (2019b). A Multi-task Approach for Disentangling Syntax and Semantics in Sentence Representations. In *Proceedings of NAACL*.

- Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., & Abbeel, P. (2017b). Variational Lossy Autoencoder. In *Proceedings of ICLR*.
- Cheng, J. & Lapata, M. (2016). Neural summarization by extracting sentences and words. In *Proceedings of ACL*.
- Cho, K., Raiko, T., Ilin, A., & Karhunen, J. (2013). A Two-Stage Pretraining Algorithm for Deep Boltzmann Machines. In *Proceedings of ICANN*.
- Cho, K., Raiko, T., Ilin, A., & Karhunen, J. (2015). How to Pretrain Deep Boltzmann Machines in Two Stages. *Artificial Neural Networks, Methods and Applications in Bio-/Neuroinformatics*, 4, 201–219.
- Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014a). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *Proceedings of Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014b). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of EMNLP*.
- Choi, J., Yoo, K. M., & goo Lee, S. (2018). Learning to Compose Task-Specific Tree Structures. In *Proceedings of AAAI*.
- Chomsky, N. (1957). *Syntactic Structures*. The Hague: Mouton and Co.
- Chung, J., Ahn, S., & Bengio, Y. (2017). Hierarchical Multiscale Recurrent Neural Networks. In *Proceedings of ICLR*.
- Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., & Bengio, Y. (2015). A Recurrent Latent Variable Model for Sequential Data. In *Proceedings of NIPS*.
- Clark, A. (2001). Unsupervised Induction of Stochastic Context Free Grammars Using Distributional Clustering. In *Proceedings of CoNLL*.
- Cohen, S. B., Gimpel, K., & Smith, N. A. (2009). Logistic Normal Priors for Unsupervised Probabilistic Grammar Induction. In *Proceedings of NIPS*.
- Cohen, S. B. & Smith, N. A. (2009). Shared Logistic Normal Distributions for Soft Parameter Tying in Unsupervised Grammar Induction. In *Proceedings of NAACL*.
- Cohen, S. B., Stratos, K., Collins, M., Foster, D. P., & Ungar, L. (2012). Spectral Learning of Latent-Variable PCFGs. In *Proceedings of ACL*.

- Cohn, T., Hoang, C. D. V., Vymolova, E., Yao, K., Dyer, C., & Haffari, G. (2016). Incorporating structural alignment biases into an attentional neural translation. *arXiv:1601.01085*.
- Collins, M. (1997). Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of ACL*.
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (almost) from Scratch. *Journal of Machine Learning Research*, 12, 2493–2537.
- Corro, C. & Titov, I. (2018). Differentiable Perturb-and-Parse: Semi-Supervised Parsing with a Structured Variational Autoencoder. *arXiv:1807.09875*.
- Cremer, C., Li, X., & Duvenaud, D. (2018). Inference Suboptimality in Variational Autoencoders. In *Proceedings of ICML*.
- Creutz, M. & Lagus, K. (2002). Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*.
- Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1), 1–38.
- Deng, Y., Kanervisto, A., Ling, J., & Rush, A. M. (2017). Image-to-Markup Generation with Coarse-to-Fine Attention. In *Proceedings of ICML*.
- Deng, Y., Kim, Y., Chiu, J., Guo, D., & Rush, A. M. (2018). Latent Alignment and Variational Attention. In *Proceedings of NeurIPS*.
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805*.
- Dieng, A. B., Wang, C., Gao, J., & Paisley, J. (2017). TopicRNN: A Recurrent Neural Network With Long-Range Semantic Dependency. In *Proceedings of ICLR*.
- Domke, J. (2012). Generic Methods for Optimization-based Modeling. In *Proceedings of AISTATS*.
- dos Santos, C. N. & Zadrozny, B. (2014). Learning Character-level Representations for Part-of-Speech Tagging. In *Proceedings of ICML*.
- Drozдов, A., Verga, P., Yadev, M., Iyyer, M., & McCallum, A. (2019). Unsupervised Latent Tree Induction with Deep Inside-Outside Recursive Auto-Encoders. In *Proceedings of NAACL*.

- Du, S. S., Zhai, X., Poczos, B., & Singh, A. (2019). Gradient Descent Provably Optimizes Over-parameterized Neural Networks. In *Proceedings of ICLR*.
- Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12.
- Durrett, G. & Klein, D. (2015). Neural CRF Parsing. In *Proceedings of ACL*.
- Dyer, C., Ballesteros, M., Ling, W., Matthews, A., & Smith, N. A. (2015). Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In *Proceedings of ACL*.
- Dyer, C., Chahuneau, V., & Smith, N. A. (2013). A Simple, Fast, and Effective Reparameterization of IBM Model 2. In *Proceedings of NAACL*.
- Dyer, C., Kuncoro, A., Ballesteros, M., & Smith, N. A. (2016). Recurrent Neural Network Grammars. In *Proceedings of NAACL*.
- Edunov, S., Ott, M., Auli, M., Grangier, D., & Ranzato, M. (2018). Classical Structured Prediction Losses for Sequence to Sequence Learning. In *Proceedings of NAACL*.
- Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, 14(2), 179–211.
- Emami, A. & Jelinek, F. (2005). A Neural Syntactic Language Model. *Machine Learning*, 60, 195–227.
- Eriguchi, A., Tsuruoka, Y., & Cho, K. (2017). Learning to Parse and Translate Improves Neural Machine Translation. In *Proceedings of ACL*.
- Finkel, J. R., Kleeman, A., & Manning, C. D. (2008). Efficient, Feature-based, Conditional Random Field Parsing. In *Proceedings of ACL*.
- Finkel, J. R., Manning, C. D., & Ng, A. Y. (2006). Solving the Problem of Cascading Errors: Approximate Bayesian Inference for Linguistic Annotation Pipelines. In *Proceedings of EMNLP*.
- Fraccaro, M., Sonderby, S. K., Paquet, U., & Winther, O. (2016). Sequential Neural Models with Stochastic Layers. In *Proceedings of NIPS*.
- Fu, H., Li, C., Liu, X., Gao, J., Celikyilmaz, A., & Carin, L. (2019). Cyclical annealing schedule: A simple approach to mitigating kl vanishing. In *Proceedings of NAACL*.
- Fu, M. C. (2006). Gradient Estimation. *Handbooks in operations research and management science*, 13, 575–616.

- Gal, Y. & Ghahramani, Z. (2016a). A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Proceedings of NIPS*.
- Gal, Y. & Ghahramani, Z. (2016b). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of ICML*.
- Ganchev, K., Graça, J., Gillenwater, J., & Taskar, B. (2010). Posterior Regularization for Structured Latent Variable Models. *Journal of Machine Learning Research*, 11, 2001–2049.
- Glasserman, P. (2013). *Monte Carlo Methods in Financial Engineering*, volume 53. Springer Science & Business Media.
- Glynn, P. (1987). Likelihood Ratio Gradient Estimation: An Overview. In *Proceedings of Winter Simulation Conference*.
- Gold, E. M. (1967). Language Identification in the Limit. *Information and Control*, 10, 447–747.
- Golland, D., DeNero, J., & Uszkoreit, J. (2012). A Feature-Rich Constituent Context Model for Grammar Induction. In *Proceedings of ACL*.
- Goodman, J. (1998). Parsing Inside-Out. *PhD thesis, Harvard University*.
- Gū, J., Shavarani, H. S., & Sarkar, A. (2018). Top-down Tree Structured Decoding with Syntactic Connections for Neural Machine Translation and Parsing. In *Proceedings of EMNLP*.
- Goyal, A., Sordoni, A., Cote, M.-A., Ke, N. R., & Bengio, Y. (2017a). Z-Forcing: Training Stochastic Recurrent Networks. In *Proceedings of NIPS*.
- Goyal, P., Hu, Z., Liang, X., Wang, C., & Xing, E. (2017b). Nonparametric Variational Auto-encoders for Hierarchical Representation Learning. In *Proceedings of ICCV*.
- Gulcehre, C., Chandar, S., Cho, K., & Bengio, Y. (2016). Dynamic Neural Turing Machine with Soft and Hard Addressing Schemes. *arXiv:1607.00036*.
- Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A. A., Visin, F., Vazquez, D., & Courville, A. (2017). PixelVAE: A Latent Variable Model for Natural Images. In *Proceedings of ICLR*.
- Guu, K., Hashimoto, T. B., Oren, Y., & Liang, P. (2017). Generating Sentences by Editing Prototypes. *arXiv:1709.08878*.

- Hale, J., Dyer, C., Kuncoro, A., & Brennan, J. R. (2018). Finding Syntax in Human Encephalography with Beam Search. In *Proceedings of ACL*.
- Havrylov, S., Kruszewski, G., & Joulin, A. (2019). Cooperative Learning of Disjoint Syntax and Semantics. In *Proceedings of NAACL*.
- Hazan, T. & Jaakkola, T. (2012). On the Partition Function and Random Maximum A-Posteriori Perturbation. In *Proceedings of ICML*.
- He, J., Neubig, G., & Berg-Kirkpatrick, T. (2018). Unsupervised Learning of Syntactic Structure with Invertible Neural Projections. In *Proceedings of EMNLP*.
- He, J., Spokoyny, D., Neubig, G., & Berg-Kirkpatrick, T. (2019). Lagging inference networks and posterior collapse in variational autoencoders. In *Proceedings of ICLR*.
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *arXiv:1512.03385*.
- He, L., Lee, K., Lewis, M., & Zettlemoyer, L. (2017). Deep semantic role labeling: What works and what’s next. In *Proceedings of ACL*.
- Hewitt, J. & Manning, C. D. (2019). A Structural Probe for Finding Syntax in Word Representations.
- Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. *arXiv:1207.0580*.
- Hinton, G. E. & van Camp, D. (1993). Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights. In *Proceedings of COLT*.
- Hjelm, R. D., Cho, K., Chung, J., Salakhutdinov, R., Calhoun, V., & Jojic, N. (2016). Iterative Refinement of the Approximate Posterior for Directed Belief Networks. In *Proceedings of NIPS*.
- Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.
- Hoffman, M. D., Blei, D. M., Wang, C., & Paisley, J. (2013). Stochastic Variational Inference. *Journal of Machine Learning Research*, 13, 1303–1347.
- Horning, J. J. (1969). A study of grammatical inference. *Stanford Computer Science Tech Report*, CS 139.
- Htut, P. M., Cho, K., & Bowman, S. R. (2018). Grammar Induction with Neural Language Models: An Unusual Replication. In *Proceedings of EMNLP*.

- Hu, Z., Yang, Z., Liang, X., Salakhutdinov, R., & Xing, E. P. (2017). Toward Controlled Generation of Text. In *Proceedings of ICML*.
- Huang, Y., Zhang, M., & Tan, C. L. (2012). Improved Constituent Context Model with Features. In *Proceedings of PACLIC*.
- Hwa, R. (1999). Supervised Grammar Induction Using Training Data with Limited Constituent Information. In *Proceedings of ACL*.
- Hwa, R. (2000). Sample Selection for Statistical Grammar Induction. In *Proceedings of EMNLP*.
- Ioffe, S. & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of ICML*.
- Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., & Kavukcuoglu, K. (2017). Decoupled Neural Interfaces using Synthetic Gradients. In *Proceedings of ICML*.
- Jain, S. & Wallace, B. C. (2019). Attention is not explanation. In *Proceedings of NAACL*.
- Jang, E., Gu, S., & Poole, B. (2017). Categorical Reparameterization with Gumbel-Softmax. In *Proceedings of ICLR*.
- Jin, L., Doshi-Velez, F., Miller, T., Schuler, W., & Schwartz, L. (2018a). Depth-bounding is Effective: Improvements and Evaluation of Unsupervised PCFG Induction. In *Proceedings of EMNLP*.
- Jin, L., Doshi-Velez, F., Miller, T., Schuler, W., & Schwartz, L. (2018b). Unsupervised Grammar Induction with Depth-bounded PCFG. In *Proceedings of TACL*.
- Johnson, M. (1998). PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24, 613–632.
- Johnson, M., Duvenaud, D. K., Wiltschko, A., Adams, R. P., & Datta, S. R. (2016). Composing Graphical Models with Neural Networks for Structured Representations and Fast Inference. In *Proceedings of NIPS*.
- Johnson, M., Griffiths, T. L., & Goldwater, S. (2007). Bayesian Inference for PCFGs via Markov chain Monte Carlo. In *Proceedings of NAACL*.
- Jordan, M., Ghahramani, Z., Jaakkola, T., & Saul, L. (1999). Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37, 183–233.

- Joshi, A. K., Levy, L. S., & Takahashi, M. (1975). Tree adjunct grammars. *Journal of Computer and System Sciences*, 10, 136–163.
- Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. In *Proceedings of ACL*.
- Kawakami, K., Dyer, C., & Blunsom, P. (2018). Unsupervised Word Discovery with Segmental Neural Language Models. *arXiv:1811.09353*.
- Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In *Proceedings of EMNLP*.
- Kim, Y., Denton, C., Hoang, L., & Rush, A. M. (2017). Structured Attention Networks. In *Proceedings of ICLR*.
- Kim, Y., Dyer, C., & Rush, A. M. (2019a). Compound Probabilistic Context-Free Grammars for Grammar Induction. In *Proceedings of ACL*.
- Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016). Character-Aware Neural Language Models. In *Proceedings of AAAI*.
- Kim, Y., Rush, A. M., Yu, L., Kuncoro, A., Dyer, C., & Melis, G. (2019b). Unsupervised Recurrent Neural Network Grammars. In *Proceedings of NAACL*.
- Kim, Y., Wiseman, S., Miller, A. C., Sontag, D., & Rush, A. M. (2018a). Semi-Amortized Variational Autoencoders. In *Proceedings of ICML*.
- Kim, Y., Wiseman, S., & Rush, A. M. (2018b). A Tutorial on Deep Latent Variable Models of Natural Language. *arXiv:1812.06834*.
- Kingma, D. P. & Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *Proceedings of ICLR*.
- Kingma, D. P., Salimans, T., & Welling, M. (2015). Variational Dropout and the Local Reparameterization Trick. In *Proceedings of NIPS*.
- Kingma, D. P., Salimans, T., & Welling, M. (2016). Improving Variational Inference with Autoregressive Flow. *arXiv:1606.04934*.
- Kingma, D. P. & Welling, M. (2014). Auto-Encoding Variational Bayes. In *Proceedings of ICLR*.
- Kitaev, N. & Klein, D. (2018). Constituency Parsing with a Self-Attentive Encoder. In *Proceedings of ACL*.
- Klein, D. & Manning, C. (2002). A Generative Constituent-Context Model for Improved Grammar Induction. In *Proceedings of ACL*.

- Klein, D. & Manning, C. D. (2003). Accurate Unlexicalized Parsing. In *Proceedings of ACL*.
- Klein, D. & Manning, C. D. (2004). Corpus-based Induction of Syntactic Structure: Models of Dependency and Constituency. In *Proceedings of ACL*.
- Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., et al. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions* (pp. 177–180).: Association for Computational Linguistics.
- Koehn, P. & Knowles, R. (2017). Six Challenges for Neural Machine Translation. *arXiv:1706.03872*.
- Krishnan, R. G., Liang, D., & Hoffman, M. (2018). On the Challenges of Learning with Inference Networks on Sparse, High-dimensional Data. In *Proceedings of AIS-TATS*.
- Krishnan, R. G., Shalit, U., & Sontag, D. (2017). Structured Inference Networks for Nonlinear State Space Models. In *Proceedings of AAAI*.
- Kuncoro, A., Ballesteros, M., Kong, L., Dyer, C., Neubig, G., & Smith, N. A. (2017). What Do Recurrent Neural Network Grammars Learn About Syntax? In *Proceedings of EACL*.
- Kuncoro, A., Dyer, C., Hale, J., Yogatama, D., Clark, S., & Blunsom, P. (2018). LSTMs Can Learn Syntax-Sensitive Dependencies Well, But Modeling Structure Makes Them Better. In *Proceedings of ACL*.
- Kuncoro, A., Dyer, C., Rimell, L., Clark, S., & Blunsom, P. (2019). Scalable Syntax-Aware Language Models Using Knowledge Distillation. In *Proceedings of ACL*.
- Kurihara, K. & Sato, T. (2006). Variational Bayesian Grammar Induction for Natural Language. In *Proceedings of International Colloquium on Grammatical Inference*.
- Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of ICML*.
- Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural architectures for named entity recognition. In *Proceedings NAACL*.
- Lari, K. & Young, S. (1990). The Estimation of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm. *Computer Speech and Language*, 4, 35–56.

- Lawson, D., Chiu, C.-C., Tucker, G., Raffel, C., Swersky, K., & Jaitly, N. (2018). Learning Hard Alignments in Variational Inference. In *Proceedings of ICASSP*.
- Le, Q. & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. In *Proceedings of ICML*.
- LeCun, Y., Simard, P., & Pearlmutter, B. (1993). Automatic Learning Rate Maximization by On-line Estimation of the Hessian’s Eigenvectors. In *Proceedings of NIPS*.
- Lee, J., Mansimov, E., & Cho, K. (2018). Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. *arXiv:1802.06901*.
- Lei, T., Barzilay, R., & Jaakkola, T. (2016). Rationalizing Neural Predictions. In *Proceedings of EMNLP*.
- Li, B., Cheng, J., Liu, Y., & Keller, F. (2019a). Dependency Grammar Induction with a Neural Variational Transition-based Parser. In *Proceedings of AAAI*.
- Li, B., He, J., Neubig, G., Berg-Kirkpatrick, T., & Yang, Y. (2019b). A surprisingly effective fix for deep latent variable modeling of text. In *Proceedings of EMNLP*.
- Li, B., Mou, L., & Keller†, F. (2019c). An Imitation Learning Approach to Unsupervised Parsing. In *Proceedings of ACL*.
- Li, Z. & Eisner, J. (2009). First- and Second-Order Expectation Semirings with Applications to Minimum-Risk Training on Translation Forests. In *Proceedings of EMNLP*.
- Liang, P., Petrov, S., Jordan, M. I., & Klein, D. (2007). The Infinite PCFG using Hierarchical Dirichlet Processes. In *Proceedings of EMNLP*.
- Lin, C.-C., Ammar, W., Dyer, C., & Levin, L. (2015). Unsupervised POS Induction with Word Embeddings. In *Proceedings of NAACL*.
- Ling, J. & Rush, A. (2017). Coarse-to-fine attention models for document summarization. In *Proceedings of the Workshop on New Frontiers in Summarization*.
- Liu, N. F., Gardner, M., Belinkov, Y., Peters, M. E., & Smith, N. A. (2019). Linguistic knowledge and transferability of contextual representations. In *Proceedings of NAACL*.
- Liu, Y., Gardner, M., & Lapata, M. (2018). Structured Alignment Networks for Matching Sentences. In *Proceedings of EMNLP*.

- Liu, Y. & Lapata, M. (2017). Learning Structured Text Representations. In *Proceedings of TACL*.
- Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of EMNLP*.
- Ma, X., Gao, Y., Hu, Z., Yu, Y., Deng, Y., & Hovy, E. (2017). Dropout with Expectation-linear Regularization. In *Proceedings of ICLR*.
- Ma, X. & Hovy, E. (2016). End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings ACL*.
- Maclaurin, D., Duvenaud, D., & Adams, R. P. (2015). Gradient-based Hyperparameter Optimization through Reversible Learning. In *Proceedings of ICML*.
- Maddison, C. J., Mnih, A., & Teh, Y. W. (2017). The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *Proceedings of ICLR*.
- Maddison, C. J., Tarlow, D., & Minka, T. (2014). A* Sampling. In *Proceedings of NIPS*.
- Maillard, J. & Clark, S. (2018). Latent Tree Learning with Differentiable Parsers: Shift-Reduce Parsing and Chart Parsing. In *Proceedings of the Workshop on the Relevance of Linguistic Structure in Neural Architectures for NLP*.
- Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19, 313–330.
- Marino, J., Yue, Y., & Mandt, S. (2018). Iterative Amortized Inference. In *Proceedings of ICML*.
- Martins, A. F. T. & Astudillo, R. F. (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proceedings of ICML*.
- Marvin, R. & Linzen, T. (2018). Targeted Syntactic Evaluation of Language Models. In *Proceedings of EMNLP*.
- Matsuzaki, T., Miyao, Y., & Tsujii, J. (2005). Probabilistic CFG with Latent Annotations. In *Proceedings of ACL*.
- Melis, G., Blundell, C., Kočiský, T., Hermann, K. M., Dyer, C., & Blunsom, P. (2018a). Pushing the Bounds of Dropout. *arXiv:1805.09208*.
- Melis, G., Dyer, C., & Blunsom, P. (2018b). On the State of the Art of Evaluation in Neural Language Models. In *Proceedings of ICLR*.

- Mensch, A. & Blondel, M. (2018). Differentiable Dynamic Programming for Structured Prediction and Attention. In *Proceedings of ICML*.
- Merialdo, B. (1994). Tagging English Text with a Probabilistic Model. *Computational Linguistics*, 20(2), 155–171.
- Merity, S., Keskar, N. S., & Socher, R. (2018). Regularizing and Optimizing LSTM Language Models. In *Proceedings of ICLR*.
- Metz, L., Poole, B., Pfau, D., & Sohl-Dickstein, J. (2017). Unrolled generative adversarial networks. In *Proceedings of ICLR*.
- Miao, Y., Grefenstette, E., & Blunsom, P. (2017). Discovering Discrete Latent Topics with Neural Variational Inference. In *Proceedings of ICML*.
- Miao, Y., Yu, L., & Blunsom, P. (2016). Neural Variational Inference for Text Processing. In *Proceedings of ICML*.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781*.
- Mikolov, T., Karafiat, M., Burget, L., Cernocky, J., & Khudanpur, S. (2010). Recurrent Neural Network Based Language Model. In *Proceedings of INTERSPEECH*.
- Mnih, A. & Gregor, K. (2014). Neural Variational Inference and Learning in Belief Networks. In *Proceedings of ICML*.
- Mnih, A. & Rezende, D. J. (2016). Variational Inference for Monte Carlo Objectives. In *Proceedings of ICML*.
- Mnih, V., Heess, N., Graves, A., & Kavukcuoglu, K. (2015). Recurrent Models of Visual Attention. In *Proceedings of NIPS*.
- Mohamed, S., Rosca, M., Figurnov, M., & Mnih, A. (2019). Monte carlo gradient estimation in machine learning. *arXiv:1906.10652*.
- Mueller, J., Gifford, D., & Jaakkola, T. (2017). Sequence to Better Sequence: Continuous Revision of Combinatorial Structures. In *Proceedings of ICML*.
- Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.
- Nallapati, R., Zhou, B., dos Santos, C., Gülçehre, Ç., & Xiang, B. (2016). Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of CoNLL*.

- Naseem, T., Chen, H., Barzilay, R., & Johnson, M. (2010). Using Universal Linguistic Knowledge to Guide Grammar Induction. In *Proceedings of EMNLP*.
- Neal, R. M. & Hinton, G. E. (1998). A New View of the EM Algorithm that Justifies Incremental, Sparse and Other Variants. *Learning in Graphical Models*.
- Neubig, G., Goldberg, Y., & Dyer, C. (2017). On-the-fly Operation Batching in Dynamic Computation Graphs. *arXiv:1705.07860*.
- Niculae, V. & Blondel, M. (2017). A Regularized Framework for Sparse and Structured Neural Attention. In *Proceedings of NIPS*.
- Niculae, V., Martins, A. F. T., Blondel, M., & Cardie, C. (2018a). SparseMAP: Differentiable Sparse Structured Inference. In *Proceedings of ICML*.
- Niculae, V., Martins, A. F. T., & Cardie, C. (2018b). Towards Dynamic Computation Graphs via Sparse Latent Structure. In *Proceedings of EMNLP*.
- Noji, H., Miyao, Y., & Johnson, M. (2016). Using Left-corner Parsing to Encode Universal Structural Constraints in Grammar Induction. In *Proceedings of EMNLP*.
- Novak, R., Auli, M., & Grangier, D. (2016). Iterative Refinement for Machine Translation. *arXiv:1610.06602*.
- Nunberg, G. (1990). *The Linguistics of Punctuation*. CSLI Publications.
- Paige, B. & Wood, F. (2016). Inference Networks for sequential Monte Carlo in Graphical Models. In *Proceedings of ICML*.
- Papandreou, G. & Yuille, A. L. (2011). Perturb-and-Map Random Fields: Using Discrete Optimization to Learn and Sample from Energy Models. In *Proceedings of ICCV*.
- Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of ICML*.
- Parikh, A. P., Cohen, S. B., & Xing, E. P. (2014). Spectral Unsupervised Parsing with Additive Tree Metrics. In *Proceedings of ACL*.
- Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural Computation*, 6(1), 147–160.
- Peng, H., Parikh, A. P., Faruqui, M., Dhingra, B., & Das, D. (2019). Text Generation with Exemplar-based Adaptive Decoding. In *Proceedings of NAACL*.
- Peng, H., Thomson, S., & Smith, N. A. (2018). Backpropagating through Structured Argmax using a SPIGOT. In *Proceedings of ACL*.

- Pereira, F. & Schabes, Y. (1992). Inside-Outside Reestimation from Partially Bracketed Corpora. In *Proceedings of ACL*.
- Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep Contextualized Word Representations. In *Proceedings of NAACL*.
- Petrov, S., Barret, L., Thibaux, R., & Klein, D. (2006). Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of ACL*.
- Pollard, C. (1984). Generalized phrase structure grammars, head grammars, and natural language. *Stanford Ph.D. thesis*, (pp. 511–546).
- Ponvert, E., Baldridge, J., & Erk, K. (2011). Simplified Unsupervised Grammar Induction from Raw Text with Cascaded Finite State Methods. In *Proceedings of ACL*.
- Press, O. & Wolf, L. (2016). Using the Output Embedding to Improve Language Models. In *Proceedings of EACL*.
- Pu, Y., Gan, Z., Henao, R., Li, C., Han, S., & Carin, L. (2017). VAE Learning via Stein Variational Gradient Descent. In *Proceedings of NIPS*.
- Rabinovich, M., Stern, M., & Klein, D. (2017). Abstract Syntax Networks for Code Generation and Semantic Parsing. In *Proceedings of ACL*.
- Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. *OpenAI blog*.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners.
- Reichart, R. & Rappoport, A. (2010). Improved Fully Unsupervised Parsing with Zoomed Learning. In *Proceedings of EMNLP*.
- Rezende, D. J. & Mohamed, S. (2015a). Variational Inference with Normalizing Flows. In *Proceedings of ICML*.
- Rezende, D. J. & Mohamed, S. (2015b). Variational Inference with Normalizing Flows. In *Proceedings of ICML*.
- Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *Proceedings of ICML*.
- Robbins, H. (1951). Asymptotically Subminimax Solutions of Compound Statistical Decision Problems. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability* (pp. 131–149).: Berkeley: University of California Press.

- Robbins, H. (1956). An Empirical Bayes Approach to Statistics. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability* (pp. 157–163).: Berkeley: University of California Press.
- Rush, A. M., Chopra, S., & Weston, J. (2015). A Neural Attention Model for Abstractive Sentence Summarization. In *Proceedings of EMNLP*.
- Sakaya, J. & Klami, A. (2017). Importance Sampled Stochastic Optimization for Variational Inference. In *Proceedings of UAI*.
- Salakhutdinov, R. & Larochelle, H. (2010). Efficient Learning of Deep Boltzmann Machines. In *Proceedings of AISTATS*.
- Salimans, T., Kingma, D., & Welling, M. (2015). Markov Chain Monte Carlo and Variational Inference: Bridging the Gap. In *Proceedings of ICML*.
- Schmidt, M., Roux, N. L., & Bach, F. (2013). Minimizing Finite Sums with the Stochastic Average Gradient. *arXiv:1309.2388*.
- Schulz, P., Aziz, W., & Cohn, T. (2018). A Stochastic Decoder for Neural Machine Translation. In *Proceedings of ACL*.
- See, A., Liu, P. J., & Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. In *Proceedings of ACL*.
- Seginer, Y. (2007). Fast Unsupervised Incremental Parsing. In *Proceedings of ACL*.
- Semeniuta, S., Severyn, A., & Barth, E. (2017). A Hybrid Convolutional Variational Autoencoder for Text Generation. *arXiv:1702.02390*.
- Sennrich, R., Haddow, B., & Birch, A. (2016). Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of ACL*.
- Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. (2017). Bi-Directional Attention Flow for Machine Comprehension. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.
- Serban, I. V., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A., & Bengio, Y. (2017). A Hierarchical Latent Variable Encoder-Decoder Model for Generating Dialogues. In *Proceedings of AAAI*.
- Serrano, S. & Smith, N. A. (2019). Is attention interpretable? In *Proceedings of ACL*.
- Shankar, S., Garg, S., & Sarawagi, S. (2018). Surprisingly Easy Hard-Attention for Sequence to Sequence Learning. In *Proceedings of EMNLP*.

- Shen, D., Zhang, Y., Henao, R., Su, Q., & Carin, L. (2018a). Deconvolutional Latent-Variable Model for Text Sequence Matching. In *Proceedings of AAAI*.
- Shen, Y., Lin, Z., Huang, C.-W., & Courville, A. (2018b). Neural Language Modeling by Jointly Learning Syntax and Lexicon. In *Proceedings of ICLR*.
- Shen, Y., Tan, S., Sordoni, A., & Courville, A. (2019). Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks. In *Proceedings of ICLR*.
- Shi, H., Mao, J., Gimpel, K., & Livescu, K. (2019). Visually Grounded Neural Syntax Acquisition. In *Proceedings of ACL*.
- Shi, H., Zhou, H., Chen, J., & Li, L. (2018). On Tree-based Neural Sentence Modeling. In *Proceedings of EMNLP*.
- Shieber, S. (1985). Evidence Against the Context-Freeness of Natural Language. *Linguistics and Philosophy*, 8, 333–343.
- Shin, B., Chokshi, F. H., Lee, T., & Choi, J. D. (2017). Classification of Radiology Reports Using Neural Attention Models. In *Proceedings of IJCNN*.
- Smith, N. A. & Eisner, J. (2004). Annealing Techniques for Unsupervised Statistical Language Learning. In *Proceedings of ACL*.
- Smith, N. A. & Eisner, J. (2005). Contrastive Estimation: Training Log-Linear Models on Unlabeled Data. In *Proceedings of ACL*.
- Snyder, B., Naseem, T., & Barzilay, R. (2009). Unsupervised Multilingual Grammar Induction. In *Proceedings of ACL*.
- Socher, R., Bauer, J., Manning, C. D., & Ng, A. Y. (2013a). Parsing with Compositional Vector Grammars. In *Proceedings of ACL*.
- Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013b). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of EMNLP*.
- Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., & Winther, O. (2016). Ladder Variational Autoencoders. In *Proceedings of NIPS*.
- Spitkovsky, V. I., Alshawi, H., & Jurafsky, D. (2012). Three Dependency-and-Boundary Models for Grammar Induction. In *Proceedings of EMNLP-CoNLL*.
- Spitkovsky, V. I., Alshawi, H., & Jurafsky, D. (2013). Breaking Out of Local Optima with Count Transforms and Model Recombination: A Study in Grammar Induction. In *Proceedings of EMNLP*.

- Srikumar, V., Kundu, G., & Roth, D. (2012). On Amortizing Inference Cost for Structured Prediction”. In *Proceedings of EMNLP*.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*.
- Steedman, M. (1987). Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5, 403–439.
- Stern, M., Andreas, J., & Klein, D. (2017). A Minimal Span-Based Neural Constituency Parser. In *Proceedings of ACL*.
- Stoyanov, V., Ropson, A., & Eisner, J. (2011). Empirical Risk Minimization of Graphical Model Parameters Given Approximate Inference, Decoding, and Model Structure. In *Proceedings of AISTATS*.
- Stratos, K. (2019). Mutual Information Maximization for Simple and Accurate Part-of-Speech Induction. In *Proceedings of NAACL*.
- Strubell, E., Verga, P., Andor, D., Weiss, D., & McCallum, A. (2018). Linguistically-Informed Self-Attention for Semantic Role Labeling. In *Proceedings of EMNLP*.
- Su, J., Wu, S., Xiong, D., Lu, Y., Han, X., & Zhang, B. (2018). Variational Recurrent Neural Machine Translation. In *Proceedings of AAAI*.
- Sutskever, I., Vinyals, O., & Le, Q. (2014). Sequence to Sequence Learning with Neural Networks. In *Proceedings of NIPS*.
- Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of ACL*.
- Tenney, I., Das, D., & Pavlick, E. (2019). Bert rediscovers the classical nlp pipeline. In *Proceedings of ACL*.
- Tomczak, J. M. & Welling, M. (2018). VAE with a VampPrior. In *Proceedings of AISTATS*.
- Tran, K. & Bisk, Y. (2018). Inducing Grammars with and for Neural Machine Translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*.
- Tran, K., Bisk, Y., Vaswani, A., Marcu, D., & Knight, K. (2016). Unsupervised Neural Hidden Markov Models. In *Proceedings of the Workshop on Structured Prediction for NLP*.

- Tu, L. & Gimpel, K. (2018). Learning Approximate Inference Networks for Structured Prediction. In *Proceedings of ICLR*.
- Tu, Z., Lu, Z., Liu, Y., Liu, X., & Li, H. (2016). Modeling Coverage for Neural Machine Translation. In *Proceedings of ACL*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. In *Proceedings of NIPS*.
- Vijay-Shanker, K. & Weir, D. J. (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27(6), 511–546.
- Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., & Hinton, G. (2015). Grammar as a Foreign Language. In *Proceedings of NIPS*.
- Vogel, S., Ney, H., & Tillmann, C. (1996). Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2* (pp. 836–841).: Association for Computational Linguistics.
- Wainwright, M. J. & Jordan, M. I. (2008). Introduction to Variational Methods for Graphical Models. *Foundations and Trends in Machine Learning*, 1, 1–103.
- Wang, P. & Blunsom, P. (2013). Collapsed Variational Bayesian Inference for PCFGs. In *Proceedings of CoNLL*.
- Wang, S. & Jiang, J. (2017). Machine comprehension using match-lstm and answer pointer. In *Proceedings of ICLR*.
- Wang, W. & Chang, B. (2016). Graph-based Dependency Parsing with Bidirectional LSTM. In *Proceedings of ACL*.
- Wang, W., Gan, Z., Wang, W., Shen, D., Huang, J., Ping, W., Satheesh, S., & Carin, L. (2018a). Topic Compositional Neural Language Model. In *Proceedings of AISTATS*.
- Wang, X., Pham, H., Yin, P., & Neubig, G. (2018b). A Tree-based Decoder for Neural Machine Translation. In *Proceedings of EMNLP*.
- Wiegrefe, S. & Pinter, Y. (2019). Attention is not not explanation. In *Proceedings of EMNLP*.
- Williams, A., Drozdov, A., & Bowman, S. R. (2018). Do Latent Tree Learning Models Identify Meaningful Structure in Sentences? In *Proceedings of TACL*.
- Williams, R. J. (1992). Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8.

- Wiseman, S. & Kim, Y. (2019). Amortized Bethe Free Energy Minimization for Learning MRFs. In *Proceedings of NeurIPS*.
- Wiseman, S., Shieber, S. M., & Rush, A. M. (2018). Learning Neural Templates for Text Generation. In *Proceedings of EMNLP*.
- Wu, S., Shapiro, P., & Cotterell, R. (2018). Hard Non-Monotonic Attention for Character-Level Transduction. In *Proceedings of EMNLP*.
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Qin Gao, K. M., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., George Kurian, N. P., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., & Dean, J. (2016). Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv:1609.08144*.
- Xiong, C., Zhong, V., & Socher, R. (2017). Dynamic coattention networks for question answering. In *Proceedings of ICLR*.
- Xu, J., Hsu, D., & Maleki, A. (2018). Benefits of Over-Parameterization with EM. In *Proceedings of NeurIPS*.
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proceedings of ICML*.
- Xue, N., Xia, F., dong Chiou, F., & Palmer, M. (2005). The Penn Chinese Treebank: Phrase Structure Annotation of a Large Corpus. *Natural Language Engineering*, 11, 207–238.
- Yang, Z., Dai, Z., Salakhutdinov, R., & Cohen, W. W. (2018). Breaking the Softmax Bottleneck: A High-Rank RNN Language Model. In *Proceedings of ICLR*.
- Yang, Z., Hu, Z., Salakhutdinov, R., & Berg-Kirkpatrick, T. (2017). Improved Variational Autoencoders for Text Modeling using Dilated Convolutions. In *Proceedings of ICML*.
- Yin, P. & Neubig, G. (2017). A Syntactic Neural Model for General-Purpose Code Generation. In *Proceedings of ACL*.
- Yin, P., Zhou, C., He, J., & Neubig, G. (2018). StructVAE: Tree-structured Latent Variable Models for Semi-supervised Semantic Parsing. In *Proceedings of ACL*.
- Yogatama, D., Blunsom, P., Dyer, C., Grefenstette, E., & Ling, W. (2017). Learning to Compose Words into Sentences with Reinforcement Learning. In *Proceedings of ICLR*.

- Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent Neural Network Regularization. *arXiv:1409.2329*.
- Zeiler, M. D. (2012). Adadelta: An Adaptive Learning Rate Method. *arXiv:1212.5701*.
- Zhang, B., Xiong, D., Su, J., Duan, H., & Zhang, M. (2016). Variational Neural Machine Translation. In *Proceedings of EMNLP*.
- Zhang, C., Butepage, J., Kjellstrom, H., & Mandt, S. (2017). Advances in Variational Inference. *arXiv:1711.05597*.
- Zhao, Y., Zhang, L., & Tu, K. (2018). Gaussian Mixture Latent Vector Grammars. In *Proceedings of ACL*.
- Zhu, C., Zhao, Y., Huang, S., Tu, K., & Ma, Y. (2017). Structured Attentions for Visual Question Answering. In *Proceedings of ICCV*.
- Zhu, X., Sobhani, P., & Guo, H. (2015). Long Short-Term Memory Over Tree Structures. In *Proceedings of ICML*.