Graphical Models with Structured Factors, Neural Factors, and Approximation-aware Training

by
Matthew R. Gormley

A dissertation submitted to Johns Hopkins University in conformity
with the requirements for the degree of Doctor of Philosophy

Baltimore, MD
October 2015

Abstract

This thesis broadens the space of rich yet practical models for structured prediction. We introduce a general framework for modeling with four ingredients: (1) latent variables, (2) structural constraints, (3) learned (neural) feature representations of the inputs, and (4) training that takes the approximations made during inference into account. The thesis builds up to this framework through an empirical study of three NLP tasks: semantic role labeling, relation extraction, and dependency parsing—obtaining state-of-the-art results on the former two. We apply the resulting *graphical models with structured and neural factors, and approximation-aware learning* to jointly model part-of-speech tags, a syntactic dependency parse, and semantic roles in a low-resource setting where the syntax is unobserved. We present an alternative view of these models as *neural networks* with a topology inspired by inference on graphical models that encode our intuitions about the data.

Keywords: Machine learning, natural language processing, structured prediction, graphical models, approximate inference, semantic role labeling, relation extraction, dependency parsing.

Acknowledgements

First, I thank my co-advisors, Jason Eisner and Mark Dredze, who always managed to align their advice exactly when it mattered and challenge me through their opposition on everything else. Jason exemplified for me how to think like a visionary and to broaden my sights as a researcher, even as we continually delved deeper into our work. Mark taught me to take a step back from my research and ambitions. He showed me how to be an empiricist, a pragmatist, and a scientist. Together, Mark and Jason demonstrated all the best qualities of advisors, teachers, and mentors. I hope that some of it rubbed off on me.

Thanks to my committee members, Benjamin Van Durme and Slav Petrov, alongside Jason Eisner and Mark Dredze. Ben frequently took on the role of both publisher and critic for my work: he advertised the real-world applications of my research and challenged me to consider the linguistic underpinnings. At every step along the way, Slav's forward-looking questions were predictive of the details that would require the most attention.

Many faculty at Johns Hopkins impacted me through teaching, conversations, and mentoring. In particular, I would like to thank those who made my experience at the Center for Language and Speech Processing (CLSP) and the Human Language Technology Center of Excellence (HLTCOE) so rich: Sanjeev Khudanpur, Matt Post, Adam Lopez, Jim Mayfield, Mary Harper, David Yarowsky, Chris Callison-Burch, and Suchi Saria. Working with other researchers taught me a lot: thanks to Spence Green, Dan Bikel, Jakob Uszkoreit, Ashish Venugopal, and Percy Liang. Emails exchanges with Jason Naradowsky, André Martins, Alexander Rush, and Valentin Spitkovsky were key for replicating prior work. Thanks to David Smith, Zhifei Li, and Veselin Stoyanov, who did work that was so complementary that we couldn't resist putting it all together.

The staff at the CLSP, the HLTCOE, and the CS Department made everything a breeze, from high performance computing to finding a classroom at the last minute—special thanks to Max Thomas and Craig Harman because good code drives research.

My fellow students and postdocs made this thesis possible. My collaborations with Mo Yu and Meg Mitchell deserve particular note. Mo taught me how to use every trick in the book, and then invent three more. Meg put up with and encouraged my incessant over-engineering that eventually led to Pacaya. To my lab mates, I can't say thank you enough: Nick Andrews, Tim Vieira, Frank Ferraro, Travis Wolfe, Jason Smith, Adam Teichart, Dingquan Wang, Veselin Stoyanov, Sharon Li, Justin Snyder, Rebecca Knowles, Nathanial Wes Filardo, Michael Paul, Nanyun Peng, Markus Dreyer, Carolina Parada, Ann Irvine, Courtney Napoles, Darcey Riley, Ryan Cotterell, Tongfei Chen, Xuchen Yao, Pushpendre Rastogi, Brian Kjersten, and Ehsan Variani.

I am indebted to my friends in Baltimore. Thanks to: Andrew for listening to my research ramblings over lunch; Alan and Nick for much needed sports for the sake of rest; the Bettles, the Kuks, and the Lofti for mealshare and more; everyone who babysat; Merv and the New Song Men's Bible Study for giving me perspective.

Thanks to *nutat* and *nunan* for teaching me to seek first Jesus in all that I do. Thanks to my *anäbixel*, *baluk*, and *ch'utin mial* for introducing me to K'iche' and giving me a second home in Guatemala. Esther, thanks—you were all the motivation I needed to finish.

To my wife, Candice Gormley: you deserve the most thanks of all. You got us through every success and failure of my Ph.D. Most importantly, you showed me how to act justly, love mercy, and walk humbly with our God.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A common tension in machine learning is the tradeoff between designing models which are *practical* to use and those which capture our *intuitions* about the underlying data. This tension is particularly salient in natural language processing (NLP). To be useful, an NLP tool must (often) process text faster than it can be spoken or written. Linguistics provides explanations of generative processes which govern that data. Yet designing models that mirror these linguistic processes would quickly lead to intractability for inference and learning. This is not just a grievance for NLP researchers: for many machine learning problems there is real-world knowledge of the data that could inform model design but practical considerations rein in our ambitions. A key goal of machine learning is to enable this use of richer models.

This thesis broadens the space of rich yet practical probabilistic models for structured prediction. We introduce a general framework for modeling with four ingredients: (1) latent variables, (2) structural constraints, (3) learned (neural) feature representations of the inputs, and (4) training that takes the approximations made during inference into account. The thesis builds up to this framework through an empirical study of three NLP tasks: semantic role labeling, relation extraction, and dependency parsing—obtaining state-of-the-art results on the former two. We apply the resulting *graphical models with structured and neural factors, and approximation-aware learning* to jointly model syntactic dependency parsing and semantic role labeling in a low-resource setting where the syntax is unobserved. We also present an alternative view of these models as *neural networks* with a topology inspired by inference on graphical models that encode our intuitions about the data.

In order to situate our contributions in the literature, we next discuss related approaches and highlight prior work that acts as critical building blocks for this thesis (Section 1.1). After stating our proposed solution (Section 1.2), we provide a succinct statement of the contributions (Section 1.3) and organization (Section 1.4) of this dissertation.

1.1 Motivation and Prior Work

In this section, we discuss the reasons behind the design of the modeling framework presented in this thesis. By considering a simple example, that is representative of many application areas in machine learning, we hope to elicit the need for latent variable modeling, structured prediction, learning with inexact inference, and neural networks. Our focus here is on the solved and open problems in these areas, leaving detailed discussions of related work to later chapters.

1.1.1 Why do we want to build rich (joint) models?

One of the major limitations to machine learning is data collection. It is expensive to obtain and just when we think we have enough, a new domain for our task—or a new task altogether—comes up. Without annotated data, one might naturally gravitate to unsupervised learning. For example, in NLP, syntactic treebanks are difficult to build, so researchers (including this one) have looked to grammar induction (the unsupervised learning of syntactic parsers) for a solution (Smith (2006) and Spitkovsky (2013) represent observable progress). Yet fully unsupervised learning has two problems:

1. It's not tuned for any downstream task. Thus, the resulting predictions may or may not be useful.

2. Usually, if you have even a very small number of training examples, you can outperform the best fully unsupervised system easily. (Often even a few handwritten rules can do better, for the case of grammar induction (Haghighi and Klein, 2006; Naseem et al., 2010; Søgaard, 2012).)

So, the question remains: how can we design high-performing models that are less reliant on hand annotated data? The solution proposed by this thesis has two related facets: First, do not throw away the idea of learning latent structure (à la grammar induction); instead build it into a larger joint model. Second, do not discard data if you have it; build a joint model that can use whatever informative data you have. Let's take an example.

> **Example:** Suppose you want to do relation extraction on weblogs. You already have data for (a) relations on weblogs, (b) syntax on newswire, and (c) named entities on broadcast news. Certainly it would be foolish to throw away datasets (b) and (c) altogether. The *usual* NLP approach is to train a pipeline of systems: (a) relation extractor, (b) parser, and (c) named entity recognizer, with features of the latter two providing information to the relation extractor. However, we don't actually believe that a parser trained on newswire knows exactly what the trees on weblogs look like. But without a *joint* model of relations, parses, and named entities there's no opportunity for feedback between the components of the pipeline. A joint model recognizes that there are *latent* trees and named entities on the weblogs; and we should use the equivalent annotations on newswire and broadcast news to influence what we believe them to be.

Should we use this fancy rich model when we have lots of supervision? The jury is still out on that one; but there are plenty of examples that suggest the gains from joint modeling may be minimal if you have lots of data (cf. Gesmundo et al. (2009), Hajič et al. (2009), and Lluís et al. (2013)). The key tradeoff is that incorporating increasingly global features leads to better models of the data, but it also makes inference more challenging. However, whether we should use a joint model when supervision is *scarce* is an open question, and this thesis begins to address it.

1.1.2 Inference with Structural Constraints

As alluded to above, increasingly rich models often lead to more expensive inference. If exact inference is too hard, can't we just rely on approximate inference? That depends on what sort of models we actually want to build, and just how fast inference needs to be. Going back to our example, if we assume that we'll be modeling syntax or semantics as latent, we'll need to encode some real-world knowledge about how they behave in the form of declarative constraints. In the language of graphical models, these constraints correspond to structured factors that express an opinion about many variables at once. The basic variants of inference for graphical models don't know how to account for these sorts of factors. But there are variants that do.

Graphical models provide a concise way of describing a probability distribution over a structured output space described by a set of variables. Recent advances in approximate inference have enabled us to consider declarative *constraints* over the variables. For MAP inference (finding the variable assignment with maximum score), the proposed methods use loopy belief propagation (Duchi et al., 2006), integer linear programming (ILP) (Riedel and Clarke, 2006; Martins et al., 2009), dual decomposition (Koo et al., 2010), or the alternating directions methods of multipliers (Martins et al., 2011b). For marginal inference (summing over variable assignments), loopy belief propagation has been employed (Smith and Eisner, 2008). Common to all but the ILP approaches is the embedding of dynamic programming algorithms (e.g. bipartite matching, forward-backward, inside-outside) within a broader coordinating framework. Even the ILP algorithms reflect the structure of the dynamic programming algorithms.

At this point, we must make a decision about what sort of inference we want to do:

- **Maximization** over the latent variables (MAP inference) sounds good if you believe that your model will have high confidence and little uncertainty about the values of those variables. But this directly contradicts the original purpose for which we set out to use the joint model: we want it to capture the aspects of the data that we're uncertain about (because we didn't have enough data to train a confident model in the first place).

- **Marginalization** fits the bill for our setting: we are unsure about a particular assignment to the variables, so each variable can sum out the uncertainty of the others. In this way, we can quantify our uncertainty about each part of the model, and allow confidence to propagate through different parts of the model. Choosing structured belief propagation (BP) (Smith and Eisner, 2008) will ensure we can do so efficiently.

Having chosen marginal inference, we turn to learning.

1.1.3 Learning under approximations

This seems like a promising direction, but there's one big problem: all of the traditional learning algorithms assume that inference is exact. The richer we make our model, the less easy exact inference will be. In practice, we often use approximate inference in place of exact inference and find the traditional learning algorithms to be effective. However, the gradients in this setting only approximate and we no longer have guarantees about the resulting learned model. Not to worry: there are some (lesser used) learning algorithms that solve exactly this problem.

- For approximate MAP inference there exists a generalization of Collins (2002)'s structured perceptron to inexact search (e.g. greedy or beam-search algorithms) (Huang et al., 2012) and its extension to hypergraphs/cube-pruning (Zhang et al., 2013).
- For marginal inference by belief propagation, there have been several approaches that compute the true gradient of an approximate model either by perturbation (Domke, 2010) or automatic-differentiation (Stoyanov et al., 2011; Domke, 2011).

At first glance, it appears as though we have all the ingredients we need: a rich model that benefits from the data we have, efficient approximate marginal inference, and learning that can handle inexact inference. Unfortunately, none of the existing approximation-aware learning algorithms work with dual decomposition (Koo et al., 2010) *or* structured BP (Smith and Eisner, 2008). (Recall that beam search, like dual decomposition, would lose us the ability to marginalize.) So we'll have to invent our own. In doing so, we will answer the question of how one does learning with an approximate marginal inference algorithm that relies on embedded dynamic programming algorithms.

1.1.4 What about Neural Networks?

If you've been following recent trends in machine learning, you might wonder why we're considering graphical models at all. During the current *re-resurgence* of neural networks, they seem to work very well on a wide variety of applications. As it turns out, we'll be able to use neural networks in our framework as well. They will be just another type of factor in our graphical models. If this hybrid approach to graphical models and neural networks sounds familiar, that's because it's been around for quite a while.

The earliest examples emphasized hybrids of hidden Markov models (HMM) and neural networks (Bengio et al., 1990; Bengio et al., 1992; Haffner, 1993; Bengio and Frasconi, 1995; Bengio et al., 1995; Bourlard et al., 1995)—recent work has emphasized their combination with energy-based models (Ning et al., 2005; Tompson et al., 2014) and with probabilistic language models (Morin and Bengio, 2005). Recent work has also explored the idea of neural factors within graphical models (Do and Artieres, 2010; Srikumar and Manning, 2014). Notably absent from this line of work are the declarative structural constraints mentioned above.

Neural networks have become very popular in NLP, but are often catered to a single task. To consider a specific example: the use of neural networks for syntactic parsing has grown increasingly prominent (Collobert, 2011; Socher et al., 2013a; Vinyals et al., 2015;

Dyer et al., 2015). These models provide a salient example of the use of learned features for structured prediction, particularly in those cases where the neural network feeds forward into a standard parsing architecture (Chen and Manning, 2014; Durrett and Klein, 2015; Pei et al., 2015; Weiss et al., 2015). However, their applicability to the broader space of structured prediction problems—beyond parsing—is limited. Again returning to our example, we are interested, by contrast, in modeling multiple linguistic strata *jointly*.

1.2 Proposed Solution

The far-reaching goal of this thesis is to better enable joint modeling of multi-faceted datasets with disjointed annotation of corpora. Our canonical example comes from NLP where we have many linguistic annotations (part-of-speech tags, syntactic parses, semantic roles, relations, etc.) spread across a variety of different corpora, but rarely all on the same sentences. A rich *joint* model of such seemingly disparate data sources would capture all the linguistic strata at once, taking our uncertainty in account over those not observed at training time. Thus we require the following:

1. Model representation that supports latent variables and declarative constraints

2. Efficient (assuredly approximate) inference

3. Learning that accounts for the approximations

4. Effective features (optionally learned) that capture the data

Our proposed solution to these problems finds its basis in several key ideas from prior work. Most notably: (1) factor graphs (Frey et al., 1997; Kschischang et al., 2001) to represent our model with latent variables and declarative constraints (Naradowsky et al., 2012a), (3) structured belief propagation (BP) (Smith and Eisner, 2008) for approximate inference, (4) empirical risk minimization (ERMA) (Stoyanov et al., 2011) and truncated message passing (Domke, 2011) for learning, and (5) either handcrafted or learned (neural network-based) features (Bengio et al., 1990). While each of these addresses one or more of our desiderata above, none of them fully satisfy our requirements. Yet, our framework, which builds on their combination, does exactly that.

In our framework, our model is defined by a factor graph. Factors express local or global opinions over subsets of the variables. These opinions can be soft, taking the form of a log-linear model for example, or can be hard, taking the form of a declarative constraint. The factor graph may contain cycles causing exact inference to be intractable in the general case. Accordingly, we perform approximate *marginal* inference by structured belief propagation, optionally embedding dynamic programming algorithms inside to handle the declarative constraint factors or otherwise unwieldy factors. We learn by maximizing an objective that is computed directly as a function of the marginals output by inference. The gradient is computed by backpropagation such that the approximations of our entire system may be taken into account.

The *icing on the cake* is that neural networks can be easily dropped into this framework as another type of factor. Notice that inference changes very little with a neural network as a factor: we simply "feed forward" the inputs through the network to get the scores of the factor. The neural network acts as an alternative differentiable scoring function for the factors, replacing the usual log-linear function. Learning is still done by backpropagation, where we conveniently already know how to backprop through the neural factor.

1.3 Contributions and Thesis Statement

Experimental:

1. **We empirically study the merits of latent-variable modeling in pipelined vs. joint training.** Prior work has introduced standalone methods for grammar induction and methods of jointly inferring a latent grammar with a downstream task. We fill a gap in the literature by comparing these two approaches empirically. We further present a new application of unsupervised grammar induction: low-resource semantic role labeling. distantly-supervised, and joint training settings.

2. **We provide additional evidence that hand-crafted and learned features are complementary.** For the task of relation extraction, we obtain state-of-the-art results using this combination—further suggesting that both tactics (learning vs. designing features) have merits.

Modeling:

3. **We introduce a new variety of hybrid graphical models and neural networks.** The novel combination of ingredients we propose includes latent variables, structured factors, and neural factors. When inference is exact, our class of models specifies a valid probability distribution over the output space. When inference is approximate, the class of models can be viewed as a form of deep neural network inspired by the inference algorithms (see *Learning* below).

4. **We present new models for grammar induction, semantic role labeling, relation extraction, and syntactic dependency parsing.** The models we develop include various combinations of the ingredients mentioned above.

Inference:

5. **We unify three forms of inference:** loopy belief propagation for graphical models, dynamic programming in hypergraphs, and feed-forward computation in neural networks. Taken together, we can view all three as the feed-forward computation of a very deep neural network whose topology is given by a particular choice of approximate probabilistic inference algorithm. Alternatively, we can understand this as a very simple extension of traditional approximate inference in graphical models with potential functions specified as declarative constraints, neural networks, and traditional exponential family functions.

Learning:

6. **We propose approximation-aware training for structured belief propagation with neural factors.** Treating our favorite algorithms as computational circuits (aka. deep networks) and running automatic differentiation (aka. backpropagation) to do end-to-end training is certainly an idea that's been around for a while (e.g. Bengio et al. (1995)). We apply this idea to models with structured and neural factors and demonstrate its effectiveness over a strong baseline. This extends prior work which focused on message passing algorithms for approximate inference with standard factors (Stoyanov et al., 2011; Domke, 2011).

7. **We introduce new training objectives for graphical models motivated by neural networks.** Viewing graphical models as a form of deep neural network naturally leads us to explore objective functions that (albeit common to neural networks) are novel to training of graphical models.

Thesis Statement We claim that the accuracy of graphical models can be improved by incorporating methods that are typically reserved for approaches considered to be distinct. First, we aim to validate that joint modeling with latent variables is effective at improving accuracy over standalone grammar induction. Second, we claim that incorporating neural networks alongside handcrafted features provides gains for graphical models. Third, taking the approximations of an entire system into account provides additional gains and can be done even with factors of many variables when they exhibit some special structure. Finally, we argue that the sum of these parts will provide new effective models for structured prediction.

1.4 Organization of This Dissertation

This primary contributions of this thesis are four content chapters: Chapters 3, 4, and 5 each explore a single extension to traditional graphical models (each with a different natural language application) and Chapter 6 combines these three extensions to show their complementarity.

- **Chapter 2:** *Background.* The first section places two distinct modeling approaches side-by-side: graphical models and neural networks. The similarities between the two are highlighted and a common notation is established. We briefly introduce the types of natural language structures that will form the basis of our application areas (deferring further application details to later chapters). Using the language of hypergraphs, we review dynamic programming algorithms catered to these structures. We emphasize the material that is essential for understanding the subsequent chapters and for differentiating our contributions.

- **Chapter 3:** *Latent Variables and Structured Factors (Semantic Role Labeling).* This chapter motivates our approach by providing an empirical contrast of three approaches to grammar induction with the aim of improving semantic role labeling. Experiments are presented on 6 languages.

- **Chapter 4:** *Neural and Log-linear Factors (Relation Extraction).* We present new approaches for relation extraction that combine the benefits of traditional feature-based log-linear models and neural networks (i.e. compositional embedding models). This combination is done at two levels: (1) by combining exponential family and neural factors and (2) through the use of the Feature-rich Compositional Embedding Model (FCM), which uses handcrafted features alongside word embeddings. State-of-the-art results are achieved on two relation extraction benchmarks.

- **Chapter 5:** *Approximation-aware Learning (Dependency Parsing).* We introduce a new learning approach for graphical models with structured factors. This method views Structured BP as defining a deep neural network and trains by backpropagation. Our approach compares favorably to conditional log-likelihood training on the task of syntactic dependency parsing—results on 19 languages are given.

- **Chapter 6:** *Graphical Models with Structured and Neural Factors.* This chapter combines all the ideas from the previous chapters to introduce graphical models with latent variables, structured factors, neural factors, and approximation-aware training. We introduce a new model for semantic role labeling and apply it in the same low-resource setting as Chapter 3.

- **Chapter 7:** *Conclusions.* This section summarizes our contributions and proposes directions for future work.

- **Appendix A:** *Engineering the System.* This appendix discusses Pacaya, an open source software framework for hybrid graphical models and neural networks of the sort introduced in Chapter 6.

1.5 Preface and Other Publications

This dissertation focuses on addressing new methods for broadening the types of graphical models for which learning and inference are practical and effective. In order to ensure that this dissertation maintains this cohesive focus, we omit some of the other research areas explored throughout the doctoral studies. Closest in relation is our work on nonconvex global optimization for latent variable models (Gormley and Eisner, 2013). This work showed that the Viterbi EM problem could be cast as a quadratic mathematical program with integer and nonlinear constraints, a relaxation of which could be solved and repeatedly tightened by the Reformulation Linearization Technique (RLT).

Other work focused on the preparation of datasets. For relation extraction, we designed a semi-automatic means of annotation: first a noisy system generates tens of thousands of pairs of entities in their sentential context that *might* exhibit a relation. Non-experts then make the simple binary decision of whether or not each annotation is correct (Gormley et al., 2010). As well, we produced one of the largest publicly available pipeline-annotated datasets in the world (Napoles et al., 2012; Ferraro et al., 2014). We also created a pipeline for automatic annotation of Chinese (Peng et al., 2015).

We also explored other NLP tasks. We introduced the task of cross-language coreference resolution (Green et al., 2012). As well we developed hierarchical Bayesian structured priors for topic modeling (Gormley et al., 2012), applied them to selectional preference (Gormley et al., 2011), and developed a new framework for topic model visualization (Snyder et al., 2013).

The Feature-rich Compositional Embedding Model (FCM) discussed in Chapter 4 was introduced jointly with with Mo Yu in our prior work (Gormley et al., 2015b)—as such, we do not regard the FCM as an independent contribution of this thesis. Also excluded is our additional related work on the FCM (Yu et al., 2014; Yu et al., 2015). Rather, the contribution of Chapter 4 is the demonstration of the complementarity of handcrafted features with a state-of-the-art neural network on two relation extraction tasks. For further study of the FCM and other compositional embedding models, we direct the reader to Mo Yu's thesis (Yu, 2015).

Finally, note that the goal of this thesis was to provide a thorough examination of the topics at hand. The reader may also be interested in our tutorial covering much of the necessary background material (Gormley and Eisner, 2014; Gormley and Eisner, 2015). Further, we release a software library with support for graphical models with structured factors, neural factors, and approximation-aware training (Gormley, 2015).

Chapter 2

Background

The goal of this section is to provide the necessary background for understanding the details of the models, inference, and learning algorithms used throughout the rest of this thesis. Since later chapters refer back to these background sections, the well-prepared reader may skim this chapter or skip it entirely in favor of the novel work presented in subsequent chapters.

2.1 Preliminaries

$\boldsymbol{x}$	The input (observation)
$\boldsymbol{y}$	The output (prediction)
$\{\boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)}\}_{d=1}^{D}$	Training instances
$\boldsymbol{\theta}$	Model parameters
$\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x})$	Feature vector
Y_i, Y_j, Y_k	Variables in a factor graph
α, β, γ	Factors in a factor graph
$\psi_\alpha, \psi_\beta, \psi_\gamma$	Potential functions for the corresponding factors
$m_{i\to\alpha}(y_i), m_{\alpha\to i}(y_i)$	Message from variable to factor / factor to variable
$b_i(y_i)$	Variable belief
$b_\alpha(\boldsymbol{y}_{\boldsymbol{\alpha}})$	Factor belief
$h_{\boldsymbol{\theta}}(\boldsymbol{x})$	Decision function
$\hat{\boldsymbol{y}}$	Prediction of a decision function
$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})$	Loss function

Table 2.1: Brief Summary of Notation

2.1.1 A Simple Recipe for Machine Learning

Here we consider a recipe for machine learning. Variants of this generic approach will be used throughout this thesis for semantic role labeling (Chapter 3), relation extraction (Chapter 4), dependency parsing (Chapter 5), and joint modeling (Chapter 6).

Suppose we are given training data $\{\boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)}\}_{d=1}^{D}$, where each $\boldsymbol{x}^{(d)}$ is an observed vector and each $\boldsymbol{y}^{(d)}$ is a predicted vector. We can encode a wide variety of data in this form such as pairs $(\boldsymbol{x}, \boldsymbol{y})$ consisting of an observed sentence and a predicted parse—or an observed image and a predicted caption. Further, suppose we are given a smaller number $D_{\text{dev}} < D$ of held out development instances $\{\boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)}\}_{d=1}^{D_{\text{dev}}}$.

A simple recipe for solving many supervised machine learning problems proceeds as follows:

1. Choose a decision function: $\hat{\boldsymbol{y}} = h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(d)})$.
2. Choose a loss function: $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}^{(d)}) \in \mathbb{R}$.
3. Initialize the model parameters at time $t = 0$ to a vector of zeros: $\boldsymbol{\theta}^{(0)} = \boldsymbol{0}$.
4. While the loss on held out development data has not converged, randomly choose a training instance d and take a small step in the direction of the gradient of the loss on d: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(d)}), \boldsymbol{y}^{(d)})$, where η_t is a learning rate parameter indicating how far to step.

If the model parameters $\boldsymbol{\theta}$ come from some high dimensional continuous space $\mathbb{R}^K$, then the fourth step above solves a continuous optimization problem. The goal of that step is to (locally) minimize the empirical risk:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.1}$$

$$\text{where } J(\boldsymbol{\theta}) = \frac{1}{D} \sum_{d=1}^{D} \ell(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(d)}), \boldsymbol{y}^{(d)}) \tag{2.2}$$

Depending on the choice of decision function h and loss function ℓ, the optimization problem may or may not be convex and piecewise constant. Regardless, we can *locally* optimize it using stochastic gradient descent, the simple first-order optimization method given in steps 3-4, which takes many small steps in the direction of the gradient for a single randomly chosen training example.

Road Map for this Section Throughout this section, we will discuss various forms for the decision function h (Section 2.2 and Section 2.3), the loss function ℓ, details about stochastic optimization and regularization (Section 2.4), other objective functions (Section 2.3.4), and how to compute gradients efficiently (Section 2.2.2). We will give special attention to graphical models (Section 2.3) in which $\boldsymbol{\theta}$ are the parameters of a probabilistic model.

2.2 Neural Networks and Backpropagation

This section describes neural networks, considering both their topology (Section 2.2.1) and how to compute derivatives of the functions they define (Section 2.2.2 and Section 2.2.3). While other more thorough treatments of neural networks can be found in the literature, we

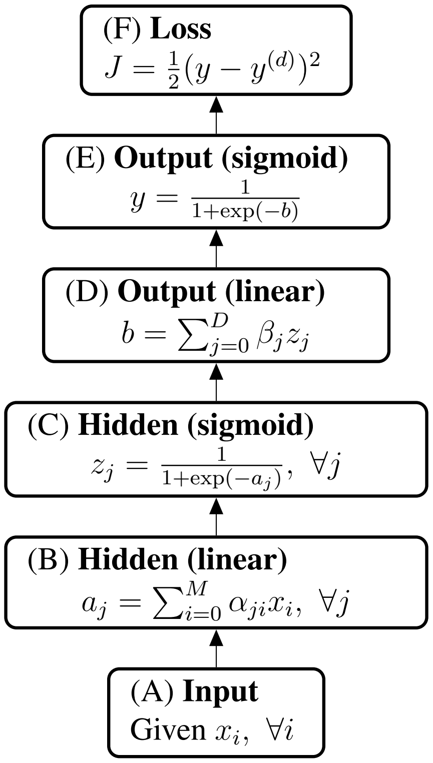

(F) **Loss**
$J = \frac{1}{2}(y - y^{(d)})^2$
(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$
(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$
(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$
(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$
(A) **Input**
Given $x_i, \ \forall i$

Figure 2.1: Feed-forward topology of a 2-layer neural network.

go into some detail here in order to facilitate connections with backpropagation through inference algorithms for graphical models—considered later in this chapter (Section 2.3.4.4).

The material presented here acts as a supplement to later uses of backpropagation such as in Chapter 4 for training of a hybrid graphical model / neural network, and in Chapter 5 and Chapter 6 for approximation-aware training.

2.2.1 Topologies

A feed-forward neural network (Rumelhart et al., 1986) defines a decision function $\boldsymbol{y} = h_{\boldsymbol{\theta}}(\boldsymbol{x})$ where $\boldsymbol{x}$ is termed the input layer and $\boldsymbol{y}$ the output layer. A feed-forward neural network has a statically defined topology. Figure 2.1 shows a simple 2-layer neural network consisting of an input layer $\boldsymbol{x}$, a hidden layer $\boldsymbol{z}$, and an output layer $\boldsymbol{y}$. In this example, the output layer is of length 1 (i.e. just a single scalar y). The model parameters of the neural network are a matrix $\boldsymbol{\alpha}$ and a vector $\boldsymbol{\beta}$.

The feed-forward computation proceeds as follows: we are given $\boldsymbol{x}$ as input (Fig. 2.1 (A)). Next, we compute an intermediate vector $\boldsymbol{a}$, each entry of which is a linear combinations of the input (Fig. 2.1 (B)). We then apply the sigmoid function $\sigma(a) = \frac{1}{1+\exp(a)}$ element-wise to obtain $\boldsymbol{z}$ (Fig. 2.1 (C)). The output layer is computed in a similar fashion, first taking a linear combination of the hidden layer to compute b (Fig. 2.1 (D)) then applying the sigmoid function to obtain the output y (Fig. 2.1 (E)). Finally we compute the loss J (Fig. 2.1 (F)) as the squared distance to the true value $y^{(d)}$ from the training data.

We refer to this topology as an arithmetic circuit. It defines both a function mapping

$\boldsymbol{x}$ to J, but also a manner of carrying out that computation in terms of the intermediate quantities $\boldsymbol{a}$, $\boldsymbol{z}$, b, y. Which intermediate quantities to use is a design decision. In this way, the arithmetic circuit diagram of Figure 2.1 is differentiated from the standard neural network diagram in two ways. A standard diagram for a neural network does not show this choice of intermediate quantities nor the form of the computations.

The topologies presented in this section are very simple. However, we will later (Chapter 5) how an entire algorithm can define an arithmetic circuit.

2.2.2 Backpropagation

The backpropagation algorithm (Rumelhart et al., 1986) is a general method for computing the gradient of a neural network. Here we generalize the concept of a neural network to include any arithmetic circuit. Applying the backpropagation algorithm on these circuits amounts to repeated application of the chain rule. This general algorithm goes under many other names: automatic differentiation (AD) in the reverse mode (*Automatic Differentiation of Algorithms: Theory, Implementation, and Application* 1991), analytic differentiation, module-based AD, autodiff, etc. Below we define a forward pass, which computes the output bottom-up, and a backward pass, which computes the derivatives of all intermediate quantities top-down.

Chain Rule At the core of the backpropagation algorithm is the chain rule. The chain rule allows us to differentiate a function f defined as the composition of two functions g and h such that $f = (g \circ h)$. If the inputs and outputs of g and h are vector-valued variables then f is as well: $h : \mathbb{R}^K \to \mathbb{R}^J$ and $g : \mathbb{R}^J \to \mathbb{R}^I \Rightarrow f : \mathbb{R}^K \to \mathbb{R}^I$. Given an input vector $\boldsymbol{x} = \{x_1, x_2, \dots, x_K\}$, we compute the output $\boldsymbol{y} = \{y_1, y_2, \dots, y_I\}$, in terms of an intermediate vector $\boldsymbol{u} = \{u_1, u_2, \dots, u_J\}$. That is, the computation $\boldsymbol{y} = f(\boldsymbol{x}) = g(h(\boldsymbol{x}))$ can be described in a feed-forward manner: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$. Then the **chain rule** must sum over all the intermediate quantities.

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}, \quad \forall i, k \tag{2.3}$$

If the inputs and outputs of f, g, and h are all scalars, then we obtain the familiar form of the chain rule:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} \tag{2.4}$$

Binary Logistic Regression Binary logistic regression can be interpreted as a arithmetic circuit. To compute the derivative of some loss function (below we use cross-entropy) with respect to the model parameters $\boldsymbol{\theta}$, we can repeatedly apply the chain rule (i.e. backpropagation). Note that the output q below is the probability that the output label takes on the value 1. y^* is the true output label. The forward pass computes the following:

$$J = y^* \log q + (1 - y^*) \log(1 - q) \tag{2.5}$$

$$\text{where } q = P_{\boldsymbol{\theta}}(Y_i = 1|\boldsymbol{x}) = \frac{1}{1 + \exp(-\sum_{j=0}^{D} \theta_j x_j)} \tag{2.6}$$

The backward pass computes $\frac{dJ}{d\theta_j} \forall j$.

Forward

$$J = y^* \log q + (1 - y^*) \log(1 - q)$$

$$q = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward

$$\frac{dJ}{dq} = \frac{y^*}{q} + \frac{(1 - y^*)}{q - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dq}\frac{dq}{da}, \; \frac{dq}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \; \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \; \frac{da}{dx_j} = \theta_j$$

2-Layer Neural Network Backpropagation for a 2-layer neural network looks very similar to the logistic regression example above. We have added a hidden layer z corresponding to the latent features of the neural network. Note that our model parameters $\boldsymbol{\theta}$ are defined as the concatenation of the vector $\boldsymbol{\beta}$ (parameters for the output layer) with the vectorized matrix $\boldsymbol{\alpha}$ (parameters for the hidden layer).

Forward

$$J = y^* \log q + (1 - y^*) \log(1 - q)$$

$$q = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dq} = \frac{y^*}{q} + \frac{(1 - y^*)}{q - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \; \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \; \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \; \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \; \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \; \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \; \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$

Notice that this application of backpropagation computes both the derivatives with respect to each model parameter $\frac{dJ}{d\alpha_{ji}}$ and $\frac{dJ}{d\beta_j}$, but also the partial derivatives with respect to each intermediate quantity $\frac{dJ}{da_j}, \frac{dJ}{dz_j}, \frac{dJ}{db}, \frac{dJ}{dy}$ and the input $\frac{dJ}{dx_i}$.

2.2.3 Numerical Differentiation

Numerical differentiation provides a convenient method for testing gradients computed by backpropagation. The *centered* finite difference approximation is:

$$\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) \approx \frac{(J(\boldsymbol{\theta} + \epsilon \cdot \boldsymbol{d}_i) - J(\boldsymbol{\theta} - \epsilon \cdot \boldsymbol{d}_i))}{2\epsilon} \tag{2.7}$$

where $\boldsymbol{d}_i$ is a 1-hot vector consisting of all zeros except for the ith entry of $\boldsymbol{d}_i$, which has value 1. Unfortunately, in practice, it suffers from issues of floating point precision. Therefore, it is typically only appropriate to use this on small examples with an appropriately chosen ϵ.

2.3 Graphical Models

This section describes some of the key aspects of graphical models that will be used throughout this thesis. This section contains many details about model representation, approximate inference, and training that form the basis for the SRL models we consider in Chapter 3. Further, these methods are considered the *baseline* against which we will compare our approximation-aware training in Chapter 5 and Chapter 6. The level of detail presented here is intended to address the interests of a practitioner who is hoping to explore these methods in their own research.

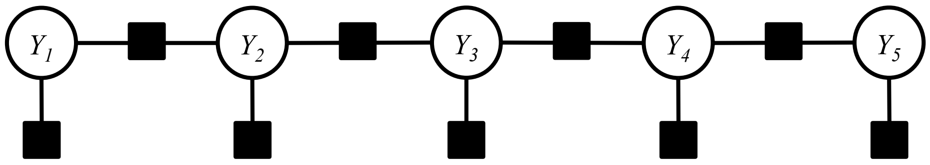
2.3.1 Factor Graphs

A graphical model defines a probability distribution $p_{\boldsymbol{\theta}}$ over a set of V predicted variables $\{Y_1, Y_2, \dots, Y_V\}$ conditioned on a set of observed variables $\{X_1, X_2, \dots, \}$. We will consider distributions of the form:

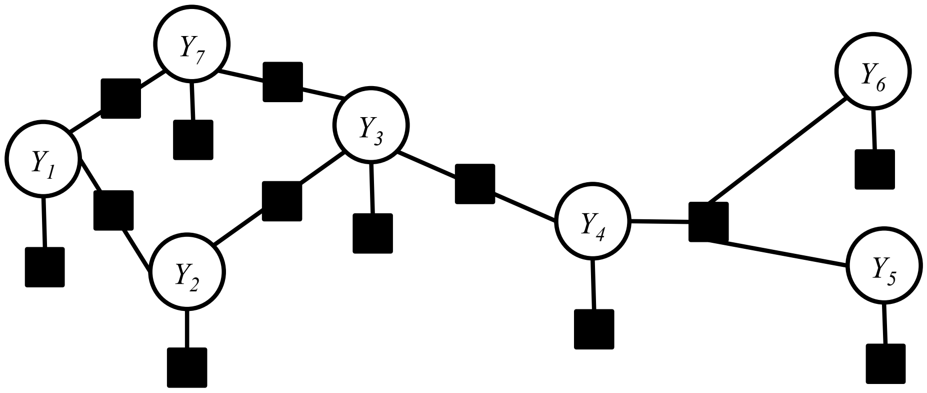
$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{\alpha \in \mathcal{F}} \psi_\alpha(\boldsymbol{y_\alpha}, \boldsymbol{x}) \tag{2.8}$$

Each $\alpha \in \mathcal{F}$ defines the indices of a subset of the variables $\alpha \subset \{1, \dots, V\}$. For each α, there is a corresponding **potential function** ψ_α, which gives a non-negative score to the variable assignments $\boldsymbol{y_\alpha} = \{y_{\alpha_1}, y_{\alpha_2}, \dots y_{\alpha_{|\alpha|}}\}$. The **partition function** $Z(\boldsymbol{x})$ is defined such that the probability distribution $p(\cdot \mid \boldsymbol{x})$ sums to one:

$$Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \prod_{\alpha} \psi_\alpha(\boldsymbol{y_\alpha}, \boldsymbol{x}) \tag{2.9}$$



(a)



(b)

Figure 2.2: Example factor graphs. The top factor graph (a) is a chain and acyclic. The bottom factor graph (b) contains cycles (i.e. it's "loopy").

For convenience, we will sometimes drop the conditioning on $\boldsymbol{x}$ when it is clear from context that the observations are available to all ψ_α, giving distributions of the form:

$$p(\boldsymbol{y}) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\boldsymbol{y_\alpha}) \tag{2.10}$$

where it is implied that the observations $\boldsymbol{x}$ are available to each of the potential functions ψ_α.

A factor graph (Frey et al., 1997; Kschischang et al., 2001) provides a visual representation for the structure of a probability distribution of the form in equation (2.8). Examples are given in Figure 2.2. Formally, a **factor graph** is a bipartite graph $\mathcal{G} = (\mathcal{V} \cup \mathcal{F}, \mathcal{E})$ comprised of a set of variable nodes $\mathcal{V}$, factor nodes $\mathcal{F}$, and edges $\mathcal{E}$. A variable $Y_i \in \mathcal{V}$ is said to have **neighbors** $\mathcal{N}(Y_i) = \{\alpha \in \mathcal{F} : i \in \alpha\}$, each of which is a factor α. Here we have overloaded α to denote both the factor node, and also the index set of its neighboring variables $\mathcal{N}(\alpha) \subset \mathcal{V}$. The graph defines a particular factorization of the distribution p_θ over variables $\boldsymbol{Y}$. The name *factor graph* highlights an important consideration throughout this thesis: how a probability distribution *factorizes* into potential functions will determine greatly the extent to which we can apply our machine learning toolbox to learn its parameters and make predictions with it.

The model form in equation (2.10) described above is sufficiently general to capture Markov random fields (MRF) (undirected graphical models), and Bayesian networks (directed graphical models)—though for the latter the potential functions ψ_α must be constrained to sum-to-one. Trained discriminatively, without such a constraint, the distribution in equation (2.8) corresponds to a conditional random field (CRF) (Lafferty et al., 2001).

2.3.2 Minimum Bayes Risk Decoding

From our earlier example, we noted that it is sometimes desirable to define a decision function $h_{\boldsymbol{\theta}}(\boldsymbol{x})$, which takes an observation $\boldsymbol{x}$ and predicts a single $\hat{\boldsymbol{y}}$. However, the graphical models we describe in this section instead define a probability distribution $p_\theta(\boldsymbol{y} \mid \boldsymbol{x})$ over the space of possible values $\boldsymbol{y}$. So how should we best select a single one?

Given a probability distribution p_θ and a loss function $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})$, a **minimum Bayes risk (MBR)** decoder returns the variable assignment $\boldsymbol{y}$ with minimum expected loss under the model's distribution (Bickel and Doksum, 1977; Goodman, 1996).

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \operatorname*{argmin}_{\hat{\boldsymbol{y}}} \; \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})] \tag{2.11}$$

$$= \operatorname*{argmin}_{\hat{\boldsymbol{y}}} \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) \tag{2.12}$$

Consider an example MBR decoder. Let ℓ be the **0-1 loss** function: $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = 1 - \mathbb{I}(\hat{\boldsymbol{y}}, \boldsymbol{y})$, where $\mathbb{I}$ is the indicator function. That is, ℓ returns loss of 0 if $\hat{\boldsymbol{y}} = \boldsymbol{y}$ and loss of 1 otherwise. Regardless of the form of the probability distribution, equation (2.12) reduces to:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \operatorname*{argmin}_{\hat{\boldsymbol{y}}} \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})(1 - \mathbb{I}(\hat{\boldsymbol{y}}, \boldsymbol{y})) \tag{2.13}$$

$$= \operatorname*{argmax}_{\hat{\boldsymbol{y}}} \; p_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}} \mid \boldsymbol{x}) \tag{2.14}$$

That is, the MBR decoder $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ will return the most probable variable assignment according to the distribution. Equation (2.14) corresponds exactly to the MAP inference problem of equation (2.18).

For other choices of the loss function ℓ, we obtain different decoders. Let our loss function be **Hamming loss**, $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_{i=1}^{V}(1 - \mathbb{I}(\hat{y}_i, y_i))$. For each variable the MBR decoder returns the value with highest marginal probability:

$$\hat{y}_i = h_{\boldsymbol{\theta}}(\boldsymbol{x})_i = \operatorname*{argmax}_{\hat{y}_i} \; p_{\boldsymbol{\theta}}(\hat{y}_i \mid \boldsymbol{x}) \tag{2.15}$$

where $p_{\boldsymbol{\theta}}(\hat{y}_i \mid \boldsymbol{x})$ is the variable marginal given in equation (2.16).

2.3.3 Approximate Inference

Given a probability distribution defined by a graphical model, there are three common inference tasks:

Marginal Inference The first task of marginal inference computes the marginals of the variables:

$$p_{\boldsymbol{\theta}}(y_i \mid \boldsymbol{x}) = \sum_{\boldsymbol{y}' : y'_i = y_i} p_{\boldsymbol{\theta}}(\boldsymbol{y}' \mid \boldsymbol{x}) \tag{2.16}$$

and the marginals of the factors

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}_\alpha \mid \boldsymbol{x}) = \sum_{\boldsymbol{y}' : \boldsymbol{y}'_\alpha = \boldsymbol{y}_\alpha} p_{\boldsymbol{\theta}}(\boldsymbol{y}' \mid \boldsymbol{x}) \tag{2.17}$$

Partition Function The second task is that of computing the partition function $Z(\boldsymbol{x})$ given by equation (2.9). Though the computation is defined as the sum over all possible assignments to the variables $\boldsymbol{Y}$, it can also be computed as a function of the variable (2.16) and factor marginals (2.17) as we will see in Section 2.3.3.3.

MAP Inference The third task computes the variable assignment $\boldsymbol{y}$ with highest probability. This is also called the *maximum a posteriori* (MAP) assignment.

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y}}{\operatorname{argmax}} \; p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \tag{2.18}$$

2.3.3.1 Belief Propagation

The belief propagation (BP) (Pearl, 1988) algorithm can be used to compute variable marginals $p_{\boldsymbol{\theta}}(y_i \mid \boldsymbol{x})$ and factor marginals $p_{\boldsymbol{\theta}}(\boldsymbol{y}_\alpha \mid \boldsymbol{x})$ when the factor graph corresponding to $p_{\boldsymbol{\theta}}$ is acyclic. BP is a message passing algorithm and defines the following update equations for **messages** from variables to factors ($i \to \alpha$) and from factors to variables ($\alpha \to i$):

$$m_{i\to\alpha}(y_i) = \frac{1}{\kappa_{i\to\alpha}} \prod_{\beta \in \mathcal{N}(i) \setminus \alpha} m_{\beta\to i}(y_i) \tag{2.19}$$

$$m_{\alpha\to i}(y_i) = \frac{1}{\kappa_{\alpha\to i}} \sum_{\boldsymbol{y_\alpha} \sim y_i} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} m_{j\to\alpha}(y_i) \tag{2.20}$$

where $\mathcal{N}(i)$ and $\mathcal{N}(\alpha)$ denote the neighbors of y_i and α respectively, and where $\boldsymbol{y_\alpha} \sim y_i$ is standard notation to indicate that $\boldsymbol{y}_\alpha$ ranges over all assignments to the variables participating in the factor α for which the ith variable has value y_i. Above, $\kappa_{i\to\alpha}$ and $\kappa_{\alpha\to i}$ are normalization constants ensuring that the vectors $\boldsymbol{m}_{i\to\alpha}$ and $\boldsymbol{m}_{\alpha\to i}$ sum to one. BP also defines update equations for **beliefs** at the variables and factors:

$$b_i(y_i) = \frac{1}{\kappa_i} \prod_{\alpha \in \mathcal{N}(i)} m_{\alpha\to i}^{(t_{\max})}(y_i) \tag{2.21}$$

$$b_\alpha(\boldsymbol{y}_\alpha) = \frac{1}{\kappa_\alpha} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{i \in \mathcal{N}(\alpha)} m_{i\to\alpha}^{(t_{\max})}(y_i) \tag{2.22}$$

where κ_i and κ_α ensure the belief vectors $\boldsymbol{b}_i$ and $\boldsymbol{b}_\alpha$ are properly normalized.

There are several aspects of the form of these update equations to notice: (1) The messages are cavity products. That is, they compute the product of all but one of the incoming messages to a variable or factor node. This is in contrast to the beliefs, which include a product of all incoming messages. (2) The message vectors $\boldsymbol{m}_{i\to\alpha}$ and $\boldsymbol{m}_{\alpha\to i}$ always define a distribution over a variable y_i regardless of whether they are sent to or from the variable y_i. (3) The update equations must be executed in some *order*, a topic we take up below.

There are two basic strategies for executing BP:

1. An **asynchronous** (serial) update order picks the next edge $e \in \mathcal{E}$, where e may be a variable-to-factor or factor-to-variable edge. It then executes the message update equations (2.19) and (2.20) for that edge so that the corresponding message vector is updated based on the current values of all the other messages.

2. By contrast, a **synchronous** (parallel) update strategy runs all the update equations at once ((2.19) and (2.20)) caching the results in temporary storage. Once the message vectors for all edges $e \in \mathcal{E}$ have been stored, it sets the current values of the messages to be the ones just computed. That is, all the messages at time t are computed from those at time $t-1$.

An update of every message constitutes an **iteration** of BP. In practice, the asynchronous approach tends to converge faster than the synchronous approach. Further, for an asynchronous order, there are a variety of methods for choosing *which* message to send next (e.g. (Elidan et al., 2006)) that can greatly speed up convergence.

The messages are said to have **converged** when they stop changing. When the factor graph is acyclic, the algorithm is guaranteed to converge after a finite number of iterations (assuming every message is sent at each iteration).

The BP algorithm described above is properly called the **sum-product BP** algorithm and performs marginal inference. Next we consider a variant for MAP inference.

Max-product BP The **max-product BP** algorithm computes the MAP assignment ((2.18)) for acyclic factor graphs. It requires only a slight change to the BP update equations given above. Specifically we replace equation (2.20) with the following:

$$m_{\alpha\to i}(y_i) = \frac{1}{\kappa_{\alpha\to i}} \max_{\boldsymbol{y}_\alpha \sim y_i} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{j\in\mathcal{N}(\alpha)\backslash i} m_{j\to\alpha}(y_i) \tag{2.23}$$

Notice that the new equation ((2.23)) is identical to sum-product version ((2.20)) except that the summation $\sum_{\boldsymbol{y}_\alpha \sim y_i}$ was replaced with a maximization $\max_{\boldsymbol{y}_\alpha \sim y_i}$. Upon convergence, the beliefs computed by this algorithm are **max-marginals**. That is, $b_i(y_i)$ is the (unnormalized) probability of the MAP assignment under the constraint $Y_i = y_i$. From the max-marginals the MAP assignment is given by:

$$y_i^* = \operatorname*{argmax}_{y_i} b_i(y_i), \ \forall i \tag{2.24}$$

2.3.3.2 Loopy Belief Propagation

Loopy belief propagation (BP) (Murphy et al., 1999) is an approximate inference algorithm for factors with cycles (i.e. "loopy" factor graphs as shown in Figure 2.2). The form of the algorithm is identical to that of Pearl (1988)'s belief propagation algorithm described in Section 2.3.3.1 except that we ignore the cycles in the factor graph. Notice that BP is fundamentally a local message passing algorithm: each message and belief is computed only as a product of (optionally) a potential function and messages that are local (i.e. being sent to) to a single variable or factor. The update equations know nothing about the cyclicity (or lack thereof) of the factor graph.

Accordingly, loopy BP runs the message update equations ((2.19) and (2.20)) using one of the update orders described in Section 2.3.3.1. The algorithm may or may not converge, accordingly it is typically run to convergence or for a maximum number of iterations, t_{max}. Upon termination of the algorithm, the beliefs are computed with the same belief update equations ((2.21) and (2.22)). Upon termination, the beliefs are empirically a good estimate of the true marginals—and are often used in place of true marginals in high-treewidth factor graphs for which exact inference is intractable. Hereafter, since it recovers the algorithm of Section 2.3.3.1 as a special case, we will use "BP" to refer to this more general loopy sum-product BP algorithm.

2.3.3.3 Bethe Free Energy

Loopy BP is also an algorithm for locally optimizing a constrained optimization problem (Yedidia et al., 2000):

$$\min \quad F_{\text{Bethe}}(\boldsymbol{b}) \tag{2.25}$$

$$\text{s.t.} \quad b_i(y_i) = \sum_{\boldsymbol{y_\alpha} \sim y_i} b_\alpha(\boldsymbol{y_\alpha}) \tag{2.26}$$

where the objective function is the **Bethe free energy** and is defined as a function of the beliefs:

$$\begin{aligned} F_{\text{Bethe}}(\boldsymbol{b}) = &\sum_\alpha \sum_{\boldsymbol{y_\alpha}} b_\alpha(\boldsymbol{y_\alpha}) \log \left[\frac{b_\alpha(\boldsymbol{y_\alpha})}{\psi_\alpha(\boldsymbol{y_\alpha})} \right] \\ &- \sum_i (n_i - 1) \sum_{y_i} b_i(y_i) \log b_i(y_i) \end{aligned}$$

where n_i is the number of neighbors of variable Y_i in the factor graph. For cyclic graphs, if loopy BP converges, the beliefs correspond to stationary points of $F_{\text{Bethe}}(\boldsymbol{b})$ (Yedidia et al., 2000). For acyclic graphs, when BP converges, the Bethe free energy recovers the negative log partition function: $F_{\text{Bethe}}(\boldsymbol{b}) = -\log Z$. This provides an effective method of computing the partition function exactly for acyclic graphs. However, in the cyclic case, the Bethe free energy also provides an (empirically) good approximation to $-\log Z$.

2.3.3.4 Structured Belief Propagation

This section describes the efficient version of belief propagation described by Smith and Eisner (2008).

The term **constraint factor** describes factors α for which some value of the potential function $\psi_\alpha(\boldsymbol{y_\alpha})$ is 0—such a factor *constrains* the variables to avoid that configuration of $\boldsymbol{y_\alpha}$ without regard to the assignment of the other variables. Notice that constraint factors are special in this regard: any potential function that returns strictly positive values could always be "outvoted" by another potential function that strongly disagrees by multiplying in a very large or very small value.

Some factor graphs include **structured factors**. These are factors whose potential function ψ_α exhibits some interesting structure. In this section, we will consider two such factors:

1. The Exactly1 factor (Smith and Eisner, 2008) (also termed the XOR factor in Martins et al. (2010b)) constrains exactly one of its binary variables to have value 1, and all the rest to have value 0.

2. The PTree factor (Smith and Eisner, 2008) is defined over a set of $O(n^2)$ binary variables that form a dependency tree over an n word sentence.

This section is about *efficiently* sending messages from structured factors to variables. That is, we will consider cases where a factor α has a very large number of neighbors $|\mathcal{N}(\alpha)|$. In these cases, the naive computation of $m_{\alpha\to i}(y_i)$ according to equation (2.20) would be prohibitively expensive (i.e. exponential in the number of neighboring variables). Smith and Eisner (2008) show how to make these computations efficient by the use of dynamic programming. This variant of efficient loopy BP with structured factors is called **structured BP**.

Smith and Eisner (2008) give two key observations that assist in these efficient computations: First, a factor has a belief about each of its variables:

$$b_\alpha(y_i) = \sum_{\boldsymbol{y_\alpha} \sim y_i} b_\alpha(\boldsymbol{y_\alpha}) \tag{2.27}$$

This is simply another variable belief computed from the factor marginal (not to be confused with $b_i(y_i)$ in equation (2.21) which has a different subscript). Second, an outgoing message from a factor is the factor's belief with the incoming message divided out:

$$m_{\alpha\to i}(y_i) = \frac{b_\alpha(y_i)}{m_{i\to\alpha}(y_i)} \tag{2.28}$$

This follows directly from the definition of the factor belief and the messages. Notice then that we need only compute the factor's beliefs about its variables $b_\alpha(y_i)$ and then we can efficiently compute the outgoing messages.

Exactly1 Factor The potential function for the Exactly1 factor is defined as:

$$\psi_{\text{Exactly1}}(\boldsymbol{y_\alpha}) = \begin{cases} 1, & \text{if } \exists \text{ exactly one } j \text{ s.t. } y_j = \textsc{on} \text{ and } y_k = \textsc{off},\ \forall k \neq j \\ 0, & \text{otherwise} \end{cases} \tag{2.29}$$

where each binary variable Y_i has domain $\{\textsc{on}, \textsc{off}\}$. We can compute the Exactly1 factor's beliefs about each of its variables efficiently. Each of the parenthesized terms below needs to be computed only once for all the variables in $\mathcal{N}(\alpha)$.

$$b_\alpha(Y_i = \textsc{on}) = \left(\prod_{j \in \mathcal{N}(\alpha)} m_{j\to\alpha}(\textsc{off}) \right) \frac{m_{i\to\alpha}(\textsc{on})}{m_{i\to\alpha}(\textsc{off})} \tag{2.30}$$

$$b_\alpha(Y_i = \textsc{off}) = \left(\sum_{j \in \mathcal{N}(\alpha)} b_\alpha(Y_j = \textsc{on}) \right) - b_\alpha(Y_i = \textsc{on}) \tag{2.31}$$

PTREE Factor The potential function for the PTREE factor is defined as:

$$\psi_\alpha(\boldsymbol{y_\alpha}) = \begin{cases} 1, & \text{if } \boldsymbol{y_\alpha} \text{ define a valid projective dependency tree} \\ 0, & \text{otherwise} \end{cases} \tag{2.32}$$

In order to compute the factor's variable belief efficiently, the first step is to utilize the fact that $\psi(\boldsymbol{y_\alpha}) \in \{0, 1\}$.

$$\Rightarrow b_\alpha(y_i) = \sum_{\substack{\boldsymbol{y_\alpha} \sim y_i, \\ \psi(y_\alpha)=1}} \prod_{j \in \mathcal{N}(\alpha)} m_{j \to \alpha}(\boldsymbol{y_\alpha}[j]) \tag{2.33}$$

$$\tag{2.34}$$

where $y_\alpha[j]$ is the value of variable Y_j according to y_α. Next given that $Y_i \in \{\text{ON}, \text{OFF}\}$, $\forall Y_i \in \mathcal{N}(\alpha)$, we have:

$$\Rightarrow b_\alpha(Y_i = \text{ON}) = \left(\prod_{j \in \mathcal{N}(\alpha)} m_{j \to \alpha}(\text{OFF}) \right) \sum_{\substack{\boldsymbol{y_\alpha} \sim y_i, \\ \psi(y_\alpha)=1}} \prod_{\substack{j \in \mathcal{N}(\alpha): \\ \boldsymbol{y_\alpha}[j]=\text{ON}}} \frac{m_{j \to \alpha}(\text{ON})}{m_{j \to \alpha}(\text{OFF})} \tag{2.35}$$

$$\text{and } \Rightarrow b_\alpha(Y_i = \text{OFF}) = \left(\sum_{\boldsymbol{y}_\alpha} b(\boldsymbol{y}_\alpha) \right) - b_\alpha(Y_i = \text{ON}) \tag{2.36}$$

The form of (2.35) exposes how an efficient dynamic programming algorithm can carry out this computation. The initial parenthetical is simply a constant that can be multiplied in at the end. The key is that the equation contains a sum over assignments $\boldsymbol{y}$, which all correspond to projective dependency trees $\sum_{\substack{\boldsymbol{y_\alpha} \sim y_i, \\ \psi(y_\alpha)=1}}$. The summands are each a product of scores, one for each edge that is included in the tree, $\prod_{\substack{j \in \mathcal{N}(\alpha): \\ \boldsymbol{y_\alpha}[j]=\text{ON}}}$. This sum over exponentially many trees has a known polynomial time solution however.

Accordingly, Smith and Eisner (2008) first run the inside-outside algorithm where the edge weights are given by the ratios of the messages to PTREE: $\frac{m^{(t)}_{i \to \alpha}(\text{ON})}{m^{(t)}_{i \to \alpha}(\text{OFF})}$. Then they multiply each resulting edge marginal given by inside-outside by the product of all the OFF messages $\pi = \prod_i m^{(t)}_{i \to \alpha}(\text{OFF})$ to get the marginal factor belief $b_\alpha(Y_i = \text{ON})$. The sum of the weights of all the trees computed by the inside algorithm can be multiplied by π to obtain the partition function $\sum_{\boldsymbol{y}_\alpha} b(\boldsymbol{y}_\alpha)$ which is used to compute $b_\alpha(Y_i = \text{OFF})$ by (2.36). Finally they divide the belief by the incoming message $m^{(t)}_{i \to \alpha}(\text{ON})$ to get the corresponding outgoing message $m^{(t+1)}_{\alpha \to i}(\text{ON})$.

2.3.4 Training Objectives

In this section, we describe several training objectives: conditional log-likelihood (Lafferty et al., 2001), empirical risk, and empirical risk minimization under approximations (Stoyanov et al., 2011). Except where it is relevant to introduce appropriate terminology, we defer any discussion of regularization until Section 2.4.1.

2.3.4.1 Conditional Log-likelihood

When we have labeled examples $\mathcal{D}_l = \{(\boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)})\}_{d=1}^{D}$, we can discriminatively train to maximize the likelihood of the latent variables, $\boldsymbol{y}$, conditioned on the observations, $\boldsymbol{x}$. This discriminative training approach is also called conditional log-likelihood (CLL) maximization and corresponds to the CRF training of Lafferty et al. (2001). The **conditional log-likelihood** is given by:

$$L(\boldsymbol{\theta}) = \sum_{d=1}^{D} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(d)} \mid \boldsymbol{x}^{(d)}) \tag{2.37}$$

where the probability is given by equation (2.8) to have the form $p(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{\alpha \in \mathcal{F}} \psi_\alpha(\boldsymbol{y}_{\boldsymbol{\alpha}}, \boldsymbol{x})$. In this section, we consider a special case of models described in Section 2.3.1, where all of the potential functions are defined so they come from the exponential family:

$$\psi_\alpha(\boldsymbol{y}_\alpha) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}_\alpha(\boldsymbol{y}_\alpha, \boldsymbol{x})), \; \forall \alpha \tag{2.38}$$

where $\boldsymbol{f}_\alpha$ is a vector-valued feature function, usually of high dimension but sparse. For factor graphs with this restriction, the CLL is:

$$L(\boldsymbol{\theta}) = \sum_{d=1}^{D} \left(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}) - \log \sum_{\boldsymbol{y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x})) \right) \tag{2.39}$$

where $\boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}) = \sum_\alpha \boldsymbol{f}_\alpha(\boldsymbol{y}_\alpha, \boldsymbol{x}))$. The derivative of the log-likelihood takes on the familiar form from CRF training,

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{d=1}^{D} \left(f_j(\boldsymbol{y}^{(d)}, \boldsymbol{x}^{(d)}) - \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x}^{(d)})}[f_j(\boldsymbol{y}, \boldsymbol{x}^{(d)})] \right) \tag{2.40}$$

$$= \sum_{d=1}^{D} \sum_{\alpha} \left(f_{\alpha,j}(\boldsymbol{y}^{(d)}, \boldsymbol{x}^{(d)}) - \sum_{\boldsymbol{y}_{\boldsymbol{\alpha}}} p_{\boldsymbol{\theta}}(\boldsymbol{y}_{\boldsymbol{\alpha}} \mid \boldsymbol{x}^{(d)}) f_{\alpha,j}(\boldsymbol{y}_{\boldsymbol{\alpha}}, \boldsymbol{x}^{(d)}) \right) \tag{2.41}$$

The first form of the derivative (2.40) shows its form to be the difference of the observed feature counts minus the expected feature counts. The second form (2.41) shows that it can be computed easily given the factor marginals from equation (2.17).

In practice, the exact marginals needed to compute this derivative can be replaced with an approximation, such as the beliefs from loopy BP (Section 2.3.3.2). This gives the gradient of a different objective function, termed the *surrogate log-likelihood* (Wainwright, 2006). We discuss this setting in greater detail in Chapter 5.

2.3.4.2 CLL with Latent Variables

Suppose instead we want to maximize the likelihood of a subset of the variables, treating the others as latent. In this case, we have a probability distribution of the form:

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}) = \sum_{\boldsymbol{z}} p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{z}) = \sum_{\boldsymbol{z}} \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\boldsymbol{y}_\alpha, \boldsymbol{z}_\alpha) \tag{2.42}$$

where $\boldsymbol{y}$ are values of the predicted variables, $\boldsymbol{z}$ are values of the latent variables, and the dependence on $\boldsymbol{x}$ is not shown. This distribution can be rewritten in terms of two partition functions:

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}) = \frac{Z(\boldsymbol{y})}{Z} \tag{2.43}$$

where

$$Z(\boldsymbol{y}) = \sum_{\boldsymbol{z}} \prod_{\alpha} \psi_\alpha(\boldsymbol{y}_\alpha, \boldsymbol{z}_\alpha) \tag{2.44}$$

$$\tag{2.45}$$

The derivative of the conditional log-likelihood in this case reduces to the following difference of expectations:

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{d=1}^{D} \left(\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot | \boldsymbol{y}^{(d)})}[f_j(\boldsymbol{y}^{(d)}, \boldsymbol{z})] - \mathbb{E}_{\boldsymbol{y},\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot,\cdot)}[f_j(\boldsymbol{y}, \boldsymbol{z})] \right) \tag{2.46}$$

$$= \sum_{d=1}^{D} \sum_{\alpha} \left(\sum_{\boldsymbol{z}_{\boldsymbol{\alpha}}} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{\boldsymbol{\alpha}} \mid \boldsymbol{y}^{(d)}) f_{\alpha,j}(\boldsymbol{y}_\alpha^{(d)}, \boldsymbol{z}_\alpha) - \sum_{\boldsymbol{y}_{\boldsymbol{\alpha}}, \boldsymbol{z}_{\boldsymbol{\alpha}}} p_{\boldsymbol{\theta}}(\boldsymbol{y}_{\boldsymbol{\alpha}}, \boldsymbol{z}_{\boldsymbol{\alpha}}) f_{\alpha,j}(\boldsymbol{y}_{\boldsymbol{\alpha}}, \boldsymbol{z}_{\boldsymbol{\alpha}}) \right) \tag{2.47}$$

where $p(\boldsymbol{z}_\beta | \boldsymbol{y})$ is the marginal distribution over $\boldsymbol{z}_\beta$ conditioned on a fixed assignment to the variables $\boldsymbol{y}$. In practice, this marginal is computed by making a copy of the original factor graph and clamping the values of $\boldsymbol{y}$, then running inference to obtain the marginals of $\boldsymbol{z}_\beta$. The other marginal $p_{\boldsymbol{\theta}}(\boldsymbol{y}_{\boldsymbol{\alpha}}, \boldsymbol{z}_{\boldsymbol{\alpha}})$ gives the distribution of the joint assignment to $\boldsymbol{y}_{\boldsymbol{\alpha}}$ and $\boldsymbol{z}_{\boldsymbol{\alpha}}$. See Sutton and McCallum (2007) for additional details about this latent variable case.

2.3.4.3 Empirical Risk Minimization

The choice to minimize empirical risk in our simple recipe from Section 2.1.1 is a well motivated one. In fact, we usually aim to find parameters θ^* that minimize expected loss on the true data distribution over sentence/parse pairs (X, Y):

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathbb{E}[\ell(h_{\boldsymbol{\theta}}(X), Y)] \tag{2.48}$$

Since the true data distribution is unknown in practice, we estimate the objective by the expected loss over the training sample, $\{(\boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)})\}_{d=1}^{D}$. This estimate is called the **empirical risk**:

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{D} \sum_{d=1}^{D} \ell(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(d)}), \boldsymbol{y}^{(d)}) \tag{2.49}$$

There are two problems associated with this objective function. First, the optimization itself can be difficult depending on the choice of loss function ℓ. The risk could be nonconvex and piecewise constant—properties which cause problems for most typical first- or second-order optimization algorithms, such as the ones we will discuss in Section 2.4. Second, if we do successfully minimize it, the model may overfit the training data. For this reason, we usually minimize the **regularized empirical risk**.

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{D} \left(r(\boldsymbol{\theta}) + \sum_{d=1}^{D} \ell(h_\theta(\boldsymbol{x}^{(d)}), \boldsymbol{y}^{(d)}) \right) \tag{2.50}$$

where $r(\boldsymbol{\theta})$ is a regularization function that discourages large (absolute values) or non-zero values in $\boldsymbol{\theta}$. Examples are given in Section 2.4.1.

2.3.4.4 Empirical Risk Minimization Under Approximations

When inference is exact and the decoder and loss function are differentiable, it is possible to do empirical risk minimization (2.49) and regularized empirical risk minimization (2.50). Sometimes the derivatives are simple enough to be computed by hand—for neural networks they are computed by backpropagation.

Stoyanov et al. (2011) and Stoyanov and Eisner (2012) introduce **empirical risk minimization under approximations (ERMA)**, which treats the entire system including approximate inference, decoding, and loss as if it were an arithmetic circuit. That arithmetic circuit (up to but not including loss) defines some decision function $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ and its derivative can be computed by backpropagation. Figure 2.3 depicts such an arithmetic circuit. For a differentiable loss, Stoyanov et al. (2011) train the system to minimize empirical risk—taking the approximations into account.

We defer a more detailed discussion of this method until Chapter 5.

2.4 Continuous Optimization

Recent advances in stochastic optimization and online learning have been critical to the recent success of large-scale machine learning. The approaches considered in this thesis have similarly benefitted from these advances. Only on *very rare* occasions is it advisable to treat optimization as a *black box* that takes a function and returns a local optimum—on the contrary, one should open the black box to know what's inside before using it blindly.

To choose an effective optimization method, we consider three desiderata: (1) efficient use of first-order gradient computations, (2) sparse updates with regularization, and (3) low bounds on regularized regret. In this section, we discuss stochastic gradient descent (SGD), mirror descent (MD) (Nemirovsky and Yudin, 1983; Beck and Teboulle, 2003), Composite Mirror Descent (COMID) (Duchi et al., 2010a), and AdaGrad (Duchi et al., 2010b; Duchi et al., 2011). Our treatment of these algorithms is very brief and primarily aims to explain how our desiderata are met by AdaGrad with Composite Mirror Descent and ℓ_2^2-regularization via lazy updates.

The methods are presented in a tutorial style. The success of this thesis certainly depends on effective algorithms for continuous optimization. However, the main contributions can certainly be understood without the details discussed here. AdaGrad will be put

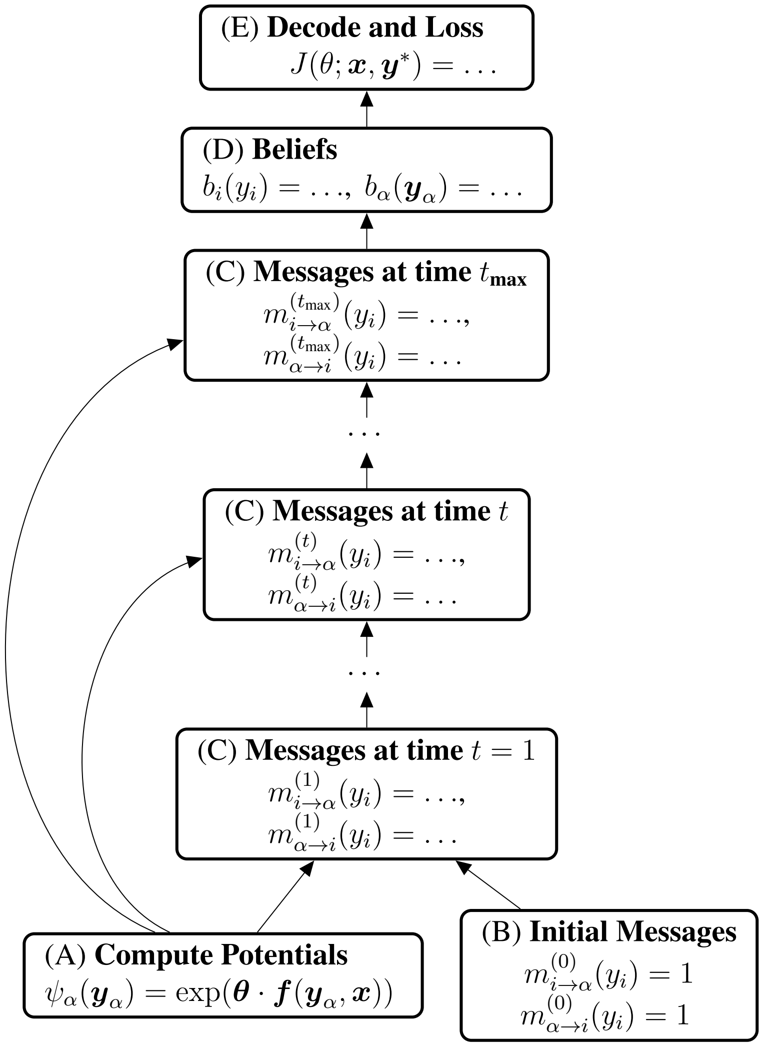

(E) Decode and Loss
$J(\theta; \boldsymbol{x}, \boldsymbol{y}^*) = \ldots$
(D) Beliefs
$b_i(y_i) = \ldots,\ b_\alpha(\boldsymbol{y}_\alpha) = \ldots$
(C) Messages at time $t_{\max}$
$m^{(t_{\max})}_{i\to\alpha}(y_i) = \ldots,$
$m^{(t_{\max})}_{\alpha\to i}(y_i) = \ldots$
(C) Messages at time t
$m^{(t)}_{i\to\alpha}(y_i) = \ldots,$
$m^{(t)}_{\alpha\to i}(y_i) = \ldots$
(C) Messages at time $t = 1$
$m^{(1)}_{i\to\alpha}(y_i) = \ldots,$
$m^{(1)}_{\alpha\to i}(y_i) = \ldots$
(A) Compute Potentials
$\psi_\alpha(\boldsymbol{y}_\alpha) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{y}_\alpha, \boldsymbol{x}))$
(B) Initial Messages
$m^{(0)}_{i\to\alpha}(y_i) = 1$
$m^{(0)}_{\alpha\to i}(y_i) = 1$

Figure 2.3: Feed-forward topology of inference, decoding, and loss according to ERMA (Stoyanov et al., 2011).

to use repeatedly for supervised and semi-supervised learning in Chapter 3, Chapter 4, Chapter 5, and Chapter 6.

2.4.1 Online Learning and Regularized Regret

Here we highlight an important connection between regularized loss minimization (the setting of the optimization problems solved in this thesis) and online learning.

Regularized Loss Minimization Throughout this thesis, our focus is generally on the problem of regularized loss minimization for some loss function $f_d(\boldsymbol{\theta})$, which is defined in terms of the training example pair $(\boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)})$. This gives us an optimization problem based on the **regularized loss** $R(\boldsymbol{\theta})$.

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} R(\boldsymbol{\theta}) \qquad \text{where } R(\boldsymbol{\theta}) = \frac{1}{D} \sum_{d=1}^{D} J_d(\boldsymbol{\theta}) + r(\boldsymbol{\theta}) \tag{2.51}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ are the model parameters, $J_d : \Theta \to \mathbb{R}$ is a loss function, and $r : \Theta \to \mathbb{R}$ is a regularization function. Θ is a convex set of possible parameters. J_d is differentiable and convex. r is convex. Example regularizers include

- ℓ_1-regularization, $r(\boldsymbol{\theta}) = \lambda ||\boldsymbol{\theta}||_1$
- ℓ_2^2-regularization, $r(\boldsymbol{\theta}) = \frac{\lambda}{2} ||\boldsymbol{\theta}||_2^2$. This is equivalent to a spherical Gaussian prior on the parameters where λ is the inverse variance. We also informally refer to this regularizer as ℓ_2 in order to better coincide with the NLP literature.

The hyperparameter λ trades off between the regularizer and loss and is typically tuned on held out data. Example loss functions include conditional log-likelihood (Section 2.3.4.1) or empirical risk (Section 2.3.4.3).

Online Learning In the **online learning** setting, we choose a sequence of parameters $\boldsymbol{\theta}^{(t)}$ for time steps $t = 1, 2, 3, \ldots$. At each time step t, an adversary gives us another loss function J_t and we receive the loss $J_t(\boldsymbol{\theta}^{(t)})$. The goal is then to ensure that the total loss up to each time step T, $\sum_{t=1}^{T} J_t(\boldsymbol{\theta}^{(t)})$, is not much worse (larger) than $\min_{\boldsymbol{\theta}} \sum_{t=1}^{T} J_t(\boldsymbol{\theta})$, which is the smallest total loss of any fixed set of parameters $\boldsymbol{\theta}$ chosen retrospectively. This is the **regret**:

$$R_T := \sum_{t=1}^{T} J_t(\boldsymbol{\theta}^{(t)}) - \min_{\boldsymbol{\theta}} \sum_{t=1}^{T} J_t(\boldsymbol{\theta}) \tag{2.52}$$

The **regularized regret** simply incorporates the regularizer r.

$$\bar{R}_T := \sum_{t=1}^{T} (J_t(\boldsymbol{\theta}^{(t)}) + r(\boldsymbol{\theta}^{(t)})) - \min_{\boldsymbol{\theta}} \sum_{t=1}^{T} (J_t(\boldsymbol{\theta}) - r(\boldsymbol{\theta})) \tag{2.53}$$

The goal is then to choose an optimization algorithm that bounds this (regularized) regret.

Connection Consider an *online learning* setting where at time step t we randomly select a training example d, defining the loss function $J_t = J_d$ to be the loss function for that training example. If our optimization algorithm bounds the regularized regret (equation (2.53)), intuitively it will also be attempting to minimize the regularized loss (equation (2.51)). Cesa-Bianchi et al. (2004) make an even stronger claim: given a bound on equation (2.53) we can obtain convergence rate and generalization bounds for equation (2.51). More to the point, if optimization algorithm has a tight bound on the regularized regret, we will converge to (local) optimum faster.

2.4.2 Online Learning Algorithms

Next we consider a sequence of four online learning algorithms—each of which extends the previous—with the goal of providing a clearer understanding of the development of the AdaGrad-COMID algorithm.

2.4.2.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) defines a simple update for each iteration.

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t(J_t'(\boldsymbol{\theta}^{(t)}) + r'(\boldsymbol{\theta}^{(t)})) \tag{2.54}$$

where $J_t'(\boldsymbol{\theta})$ is the gradient of J_t or one of its subgradients at point $\boldsymbol{\theta}$, and $r'(\boldsymbol{\theta})$ is equivalently a gradient or subgradient of $r(\boldsymbol{\theta})$. Note that we have departed from the notation of $\nabla J_t(\boldsymbol{\theta})$ for gradients, used elsewhere in this thesis, for clarity in the introduction of subsequent optimization algorithms. Intuitively, SGD takes a small step in the direction of the gradient of the regularized loss. Typically, to ensure convergence, the learning rate η_t is chosen to decay over time.

2.4.2.2 Mirror Descent

Let $\phi_t = J_t + r$ denote the sum of the loss function and regularizer at time t. Intuitively, the Mirror Descent algorithm (Nemirovsky and Yudin, 1983; Beck and Teboulle, 2003) update given below minimizes a linear approximation of the function ϕ_t at the current parameters $\boldsymbol{\theta}^{(t)}$ while ensuring that the next $\boldsymbol{\theta}^{(t+1)}$ is close to $\boldsymbol{\theta}^{(t)}$. Hereafter, for vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, we use $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ to denote their dot product. The update for Mirror Descent is:

$$\boldsymbol{\theta}^{(t+1)} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} \eta \left\langle \phi_t'(\boldsymbol{\theta}^{(t)}), \boldsymbol{\theta} - \boldsymbol{\theta}^{(t)} \right\rangle + B_\psi(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) \tag{2.55}$$

$$= \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} \eta \left\langle J_t'(\boldsymbol{\theta}^{(t)}) + r'(\boldsymbol{\theta}^{(t)}), \boldsymbol{\theta} - \boldsymbol{\theta}^{(t)} \right\rangle + B_\psi(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) \tag{2.56}$$

where η is a learning rate parameter, ϕ_t' is a (sub)gradient of ϕ_t, B_ψ is a Bregman divergence, and ψ is a carefully chosen function (more discussion below). The Bregman divergence for ψ is defined as:

$$B_\psi(\boldsymbol{w}, \boldsymbol{v}) = \psi(\boldsymbol{w}) - \psi(\boldsymbol{v}) - \langle \psi'(\boldsymbol{v}), \boldsymbol{w} - \boldsymbol{v} \rangle \tag{2.57}$$

where ψ' is the gradient of ψ. Duchi et al. (2010a) require that ψ have two properties: (a) continuously differentiable, and (b) α-strongly convex with respect to a norm $||\cdot||$ on the set of possible model parameters Θ. An example of such a function would be $\psi(\boldsymbol{w}) = \frac{1}{2}||w||_2^2$.

2.4.2.3 Composite Objective Mirror Descent

Composite objective mirror descent (COMID) (Duchi et al., 2010a) uses the following update.

$$\boldsymbol{\theta}^{(t+1)} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} \eta \left\langle J_t'(\boldsymbol{\theta}^{(t)}), \boldsymbol{\theta} - \boldsymbol{\theta}^{(t)} \right\rangle + \eta r(\boldsymbol{\theta}) + B_\psi(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) \tag{2.58}$$

This update is identical to that of Mirror Descent in equation (2.56), except that $r(\boldsymbol{\theta})$ is not linearized, but instead included directly in the minimization. Duchi et al. (2010a) give a $O(\sqrt{T})$ bound for the regret in the general case, and a $O(\log T)$ bound when $J_t(\boldsymbol{\theta}) + r(\boldsymbol{\theta})$ is strongly convex. Note that we have not yet specified exactly *how* one would compute the argmin above. For many choices of $r(\boldsymbol{\theta})$ and $\psi(\boldsymbol{w})$, this update has a closed form. We will give such a derived algorithm for AdaGrad-COMID below.

2.4.2.4 AdaGrad

The AdaGrad family of algorithms (Duchi et al., 2010b; Duchi et al., 2011) is defined in two forms. The first is based on Composite Objective Mirror Descent, and is our focus in this section. The second is derived similarly from Regularized Dual Averaging (Xiao, 2009), though we do not describe it here. The updates for both cases are defined for a very careful choice of ψ, which is adapted over time such that it is sensitive to the data. AdaGrad-COMID starts with the update from equation (2.58). The first change is that it defines a different ψ for each time step t. That is we replace ψ in equation (2.58) with ψ_t. The key contribution of AdaGrad is defining the proximal functions ψ_t to be the squared Mahalanobis norm:

$$\psi_t(\boldsymbol{\theta}) = \frac{1}{2}\langle \boldsymbol{\theta}, H_t \boldsymbol{\theta} \rangle \tag{2.59}$$

where H_t is a diagonal matrix defined such that each $H_{t,i,i} = \delta + \sqrt{\sum_{s=1}^{t} (f'_s(\boldsymbol{\theta})_i)^2}$ is a smoothed version of the square root of the sum of the squares of the ith element of the gradient over all time steps up to t. With this definition of ψ_t, the update in equation (2.58) can then be simplified to:

$$\boldsymbol{\theta}^{(t+1)} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} \left\langle \eta J'_t(\boldsymbol{\theta}^{(t)}) - H_t\boldsymbol{\theta}^{(t)}, \boldsymbol{\theta} \right\rangle + \eta r(\boldsymbol{\theta}) + \frac{1}{2}\langle \boldsymbol{\theta}, H_t\boldsymbol{\theta}\rangle \tag{2.60}$$

Intuitively, large partial derivatives along dimension i will lead to smaller (more conservative) steps in that dimension.

Derived Algorithms for AdaGrad-COMID Finally, the derived algorithms for AdaGrad-COMID for the regularizers from Section 2.4.1, can be given as closed form updates to the parameters. For the ℓ_1-regularizer, $r(\boldsymbol{\theta}) = \lambda||\boldsymbol{\theta}||_1$, we have the following update.

$$\theta_i^{(t+1)} = \text{sign}\left(\theta_i^{(t)} - \frac{\eta}{H_{t,i,i}} g_{t,i}\right)\left[\left|\theta_i^{(t)} - \frac{\eta}{H_{t,i,i}} g_{t,i}\right| - \frac{\lambda\eta}{H_{t,i,i}}\right]_+ \tag{2.61}$$

where $[x]_+ = \max(0, x)$, $\text{sign}(x)$ is the sign of x, and $g_{t,i}$ is shorthand for the ith element of $f'_t(\boldsymbol{\theta})$.

For the ℓ_2^2-regularizer, $r(\boldsymbol{\theta}) = \frac{\lambda}{2}||\boldsymbol{\theta}||_2^2$, we have the following update.

$$\theta_i^{(t+1)} = \frac{H_{t,i,i}\theta_i^{(t)} - \eta g_{t,i}}{\eta\lambda\delta + H_{t,i,i}} \tag{2.62}$$

where the hyperparameter δ helps deal with the initially noisy values in $H_{t,i,i}$ and typically takes a small positive value ≤ 1.

Regret Bound The same $O(\sqrt{T})$ and $O(\log T)$ bounds apply to AdaGrad-COMID because it is just a special case of the COMID algorithm. However, Duchi et al. (2011) further prove that the bound on the regret they obtain is as good as the best possible choice for ψ in hindsight.

AdaGrad-COMID with Lazy Updates Of importance to this work is the fact that these algorithms can be modified to support lazy updates of the model parameters (Duchi et al., 2010b; Duchi et al., 2011). This is important since the gradients we compute $J'_t(\boldsymbol{\theta})$ are based on a single training example d and are therefore very sparse (e.g. only a few thousand parameters out of tens of millions). However, due to the regularizer *every* parameter θ_i would be updated at each time step.

In the *lazy-updates* version of AdaGrad-COMID (Duchi et al., 2010b; Duchi et al., 2011), the update is only applied in equation (2.62) to those parameters $\boldsymbol{\theta}_i$ where the ith value in $J'_t(\boldsymbol{\theta})$ is non-zero. For all the other parameters the update $\theta_i^{(t+1)} = \theta_i^{(t)}$ is used. For model parameter θ_i, let t_0 denote the last time step at which the ith value of the gradient $J'_{t_0}(\boldsymbol{\theta})$ was non-zero. Then we can lazily compute $\theta_i^{(t)}$ from $\theta_i^{(t_0)}$ as below:

$$\theta_i^{(t)} = \theta_i^{(t_0)} \left(\frac{H_{t,i,i}}{\eta\lambda\delta + H_{t_0,i,i}} \right)^{(t-t_0)} \tag{2.63}$$

2.4.2.5 Parallelization over Mini-batches

Above, we assumed that $J_t(\boldsymbol{\theta}) = J_d(\boldsymbol{\theta})$ for a single randomly chosen example d from our training dataset. Alternatively, we can let J_t to be the average over a randomly chosen mini-batch of examples, $\{d_1, d_2, \ldots, d_K\}$: $J_t(\boldsymbol{\theta}) = \frac{1}{K}\sum_{k=1}^{K} J_{d_k}(\boldsymbol{\theta})$. If we view the training example d and the minibatch $\{d_1, d_2, \ldots, d_K\}$ as random variables, then the expected values of either gradient is that of the full batch gradient:

$$\mathbb{E}[J_d(\boldsymbol{\theta})] = \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} J_{d_k}(\boldsymbol{\theta})\right] = \frac{1}{D}\sum_{d=1}^{D} J_d(\boldsymbol{\theta}) \tag{2.64}$$

Often, it is preferable to use mini-batches because the gradients are a lower variance estimate of the full batch gradient. In this thesis, we prefer the mini-batch approach because it allows for a trivial method of parallelization during training: each gradient $J_{d_k}(\boldsymbol{\theta})$ can be computed independently on a separate thread with shared-memory access to the model parameters. Whenever the gradient computation dominates the computation time, this can be a very effective method of parallelization. This is the case in the models we consider for which feature extraction and inference are typically more computationally expensive than the other non-parallelized SGD computations. In practice, we typically choose the mini-batch size K to be 2-3 times the number of threads available.

Chapter 3

Latent Variables and Structured Factors

The primary goal of this chapter[1]—within the broader context of the thesis—is to motivate the use of joint modeling in low-resource settings. Our focus is the interface of syntax and semantics. To that end, we choose a specific task, semantic role labeling (SRL), for which there is strong evidence that additional structure, in the form of syntactic dependencies, is highly informative.

A possible criticism of this choice is that SRL does not represent an end-task. Ideally, to establish the dominance of joint modeling in the low-resource setting, we would pick a field such as computer vision, and jointly model all the tasks that field believes to be important and relevant—or at least as many as the field has data for. Here we only aim to motivate such a framework for joint modeling. We choose the task of low-resource SRL for several reasons:

1. A wealth of prior work has been invested into building state-of-the-art models without fancy new machinery (e.g. neural nets). Later in this thesis will explore such techniques (Chapter 4 and Chapter 6), but here we restrict to simple log-linear models.

2. SRL permits us to define a joint model of syntax/semantics for which *exact* inference is possible. Though we later address the issue of learning with approximate inference (Chapter 5), we can focus here on the simpler case of maximum likelihood estimation with exact inference.

3. Third, the question of whether joint modeling is beneficial for the high-resource setting has already been studied. Specifically, the results from the CoNLL-2009 shared task (Hajič et al., 2009) and some subsequent work (Lluís et al., 2013) demonstrated that the tradeoff of a richer model with more challenging inference vs. the rich features of a pipeline but with easy inference remains unclear. We do not seek to clarify this tradeoff for the high-resource setting, but instead aim to demonstrate that the benefits of joint modeling in the *low-resource* SRL setting are much clearer.

Note that the low-resource setting is one that *has* been explored before (Boxwell et al., 2011; Naradowsky et al., 2012a). However, the empirical studies to date leave our question

[1]A previous version of this work was presented in Gormley et al. (2014).

unanswered: is joint modeling worth it? This is because prior work did not include a controlled comparison of joint and pipelined systems in the low-resource setting.

We begin to address this question by exploring the extent to which high-resource manual annotations such as treebanks are necessary for the task of semantic role labeling (SRL). We examine how performance changes without syntactic supervision, comparing both *joint* and *pipelined* methods to induce latent syntax. This work highlights a new application of unsupervised grammar induction and demonstrates several approaches to SRL in the absence of supervised syntax. Our best models obtain competitive results in the high-resource setting and state-of-the-art results in the low-resource setting, reaching 72.48% F1 averaged across languages.

3.1 Introduction

The goal of semantic role labeling (SRL) is to identify predicates and arguments and label their semantic contribution in a sentence. Such labeling defines *who* did *what* to *whom*, *when*, *where* and *how*. For example, in the sentence “The kids ran the marathon”, *ran* assigns a role to *kids* to denote that they are the runners; and a distinct role to *marathon* since it denotes the type of event in which they are participating. By contrast, the sentence “The kids ran the horse around the track” assigns a different semantic role to *kids* even though it remains in the syntactic subject position.

Models for SRL have increasingly come to rely on an array of NLP tools (e.g., parsers, lemmatizers) in order to obtain state-of-the-art results (Björkelund et al., 2009; Zhao et al., 2009). Each tool is typically trained on hand-annotated data, thus placing SRL at the end of a very high-resource NLP pipeline. However, richly annotated data such as that provided in parsing treebanks is expensive to produce, and may be tied to specific domains (e.g., newswire). Many languages do not have such supervised resources (*low-resource languages*), which makes exploring SRL cross-linguistically difficult.

In this work, we explore models that minimize the need for high-resource supervision. We examine approaches in a **joint** setting where we marginalize over latent syntax to find the optimal semantic role assignment and a **pipeline** setting where we first induce an unsupervised grammar. We find that the joint approach is a viable alternative for making reasonable semantic role predictions, outperforming the pipeline models. These models can be effectively trained with access to only SRL annotations, and mark a state-of-the-art contribution for low-resource SRL.

To better understand the effect of the low-resource grammars and features used in these models, we further include comparisons with (1) models that use higher-resource versions of the same features; (2) state-of-the-art high resource models; and (3) previous work on low-resource grammar induction. This chapter makes several experimental and modeling contributions, summarized below.

Experimental contributions:

- Comparison of pipeline and joint models for SRL.
- Subtractive experiments that consider the removal of supervised data.

- Analysis of the induced grammars in unsupervised, distantly-supervised, and joint training settings.

Modeling contributions:

- Simpler joint CRF for syntactic and semantic dependency parsing than previously reported.
- New application of unsupervised grammar induction: low-resource SRL.
- Constrained grammar induction using SRL for distant-supervision.
- Use of Brown clusters in place of POS tags for low-resource SRL.

The pipeline models are introduced in Sections 3.2.1 and 3.2.2 and jointly-trained models for syntactic and semantic dependencies (similar in form to Naradowsky et al. (2012a)) are introduced in Section 3.2.3. In the pipeline models, we develop a novel approach to unsupervised grammar induction and explore performance using SRL as distant supervision. The joint models use a non-loopy conditional random field (CRF) with a global factor constraining latent syntactic edge variables to form a tree. Efficient exact marginal inference is possible by embedding a dynamic programming algorithm within belief propagation as in Smith and Eisner (2008).

The joint model can not efficiently incorporate the full rich feature set used by the pipeline model. Despite this shortcoming, the joint models best pipeline-trained models for state-of-the-art performance in the low-resource setting (Section 3.5.2). When the models have access to observed syntactic trees, they achieve near state-of-the-art accuracy in the high-resource setting on some languages (Section 3.5.1).

Examining the learning curve of the joint and pipeline models in two languages demonstrates that a small number of labeled SRL examples may be essential for good end-task performance, but that the choice of a good model for grammar induction has an even greater impact.

3.2 Approaches

In this section, we introduce two low-resource pipelines (Section 3.2.1) and (Section 3.2.2) and a joint model (Section 3.2.3). We also describe a supervised pipeline which acts as a skyline (Section 3.2.4). Finally, we describe the features (Section 3.2.5) that are used by all of these approaches.

3.2.1 Pipeline Model with Unsupervised Syntax

Typical SRL systems are trained following a pipeline where the first component is trained on supervised data, and each subsequent component is trained using the 1-best output of the previous components. A typical pipeline consists of a POS tagger, a dependency parser, and semantic role labeler. In a high resource setting, each of these stages of the

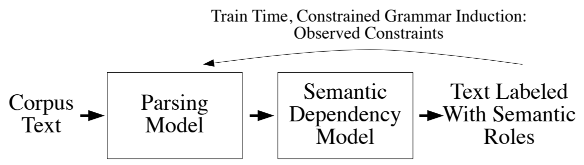

Train Time, Constrained Grammar Induction:
Observed Constraints
Corpus Text
Parsing Model
Semantic Dependency Model
Text Labeled With Semantic Roles

Figure 3.1: Pipeline approach to SRL. In this simple pipeline, the first stage syntactically parses the corpus, and the second stage predicts semantic predicate-argument structure for each sentence using the labels of the first stage as features. In our *low-resource* pipelines, we assume that the syntactic parser is given no labeled parses—however, it may optionally utilize the semantic parses as distant supervision. Our experiments also consider 'longer' pipelines that include earlier stages: a morphological analyzer, POS tagger, lemmatizer.

pipeline is trained on annotated data. In this section, we remove the need for a supervised tagger and parser. We introduce a pipeline consisting of an unsupervised tagger (Section 3.2.1.1), an unsupervised parser (Section 3.2.1.2), and a supervised semantic role labeler (Section 3.2.1.3).

3.2.1.1 Brown Clusters

The first stage of our pipeline is an unsupervised tagger. Specifically, we use fully unsupervised Brown clusters (Brown et al., 1992) in place of POS tags. Brown clusters have been used to good effect for various NLP tasks such as named entity recognition (Miller et al., 2004) and dependency parsing (Koo et al., 2008; Spitkovsky et al., 2011).

The clusters are formed by a greedy hierarchical clustering algorithm that finds an assignment of words to classes by maximizing the likelihood of the training data under a latent-class bigram model. Each word type is assigned to a fine-grained cluster at a leaf of the hierarchy of clusters. Each cluster can be uniquely identified by the path from the root cluster to that leaf. Representing this path as a bit-string (with 1 indicating a left and 0 indicating a right child) allows a simple coarsening of the clusters by truncating the bit-strings. We train 1000 Brown clusters for each of the CoNLL-2009 languages on Wikipedia text.[2] We restrict the vocabulary for each language to the 300,000 most common unigrams. The open source implementation of Liang (2005) is used for training.

3.2.1.2 Unsupervised Grammar Induction

The second stage of the pipeline is an unsupervised syntactic parser. Here we perform grammar induction by *fully unsupervised* Viterbi EM training of the Dependency Model with Valence (DMV) (Klein and Manning, 2004), with uniform initialization of the model parameters. We define the DMV such that it generates sequences of word classes: either POS tags or Brown clusters as in Spitkovsky et al. (2011). The DMV is a simple generative model for projective dependency trees. Children are generated recursively for each node. Conditioned on the parent class, the direction (right or left), and the current valence (first

[2]The Wikipedia text was tokenized for Polyglot (Al-Rfou' et al., 2013): `http://bit.ly/embeddings`

child or not), a coin is flipped to decide whether to generate another child; the distribution over child classes is conditioned on only the parent class and direction. Spitkovsky et al. (2010a) show that Viterbi (hard) EM training of the DMV with simple uniform initialization of the model parameters yields higher accuracy models than standard soft-EM training. In Viterbi EM, the E-step finds the maximum likelihood corpus parse given the current model parameters. The M-step then finds the maximum likelihood parameters given the corpus parse. We utilize this approach to produce unsupervised syntactic features for the SRL task.

We follow Spitkovsky et al. (2010a) by starting with an E-step where the model parameters are uniformly initialized. Concurrently, Cohen and Smith (2010) observed that starting with an M-step where the trees are chosen uniformly at random is also effective. For the approach we take, ties must be broken randomly in the M-step parser. Otherwise, undesirable bias may creep in during the first M-step.[3] Again following Spitkovsky et al. (2010a), we break ties within each chart cell. While this does not perfectly sample from the set of maximum likelihood trees, we found it to be empirically effective and much simpler than the algorithm required for breaking ties by sampling uniformly among trees.

3.2.1.3 Semantic Dependency Model

The third stage of the pipeline is a supervised SRL system that uses the 1-best output of the previous two unsupervised stages (Brown clusters and DMV) as input. Semantic role labeling can be cast as a structured prediction problem where the structure is a labeled semantic dependency graph. We define a conditional random field (CRF) (Lafferty et al., 2001) for this task. We describe the model here as a factor graph, as discussed in Section 2.3.1. Because each word in a sentence may be in a semantic relationship with any other word (including itself), a sentence of length n has n^2 possible edges. We define a single L+1-ary variable for each edge, whose value can be any of L semantic labels or a special label indicating there is no predicate-argument relationship between the two words. In this way, we jointly perform *identification* (determining whether a semantic relationship exists) and *classification* (determining the semantic label). We include one unary factor for each variable.

We optionally include additional variables that perform word sense disambiguation for each predicate. Each has a unary factor and is completely disconnected from the semantic edge (similar to Naradowsky et al. (2012a)). These variables range over all the predicate senses observed in the training data for the *lemma* of that predicate. This effectively treats the predicate sense prediction problem as a separate task. Including it allows us to compare with prior work on the standard CoNLL-2009 benchmark.

3.2.2 Pipeline Model with Distantly-Supervised Syntax

Our second pipeline model is identical in form to the one presented in Section 3.2.1 except that we replace the second stage: instead of *unsupervised* grammar induction as in Section 3.2.1.2, we train the parser using the semantic roles as *distant supervision*.

[3]This was observed experimentally and resolved via personal correspondence with the first author of Spitkovsky et al. (2010a).

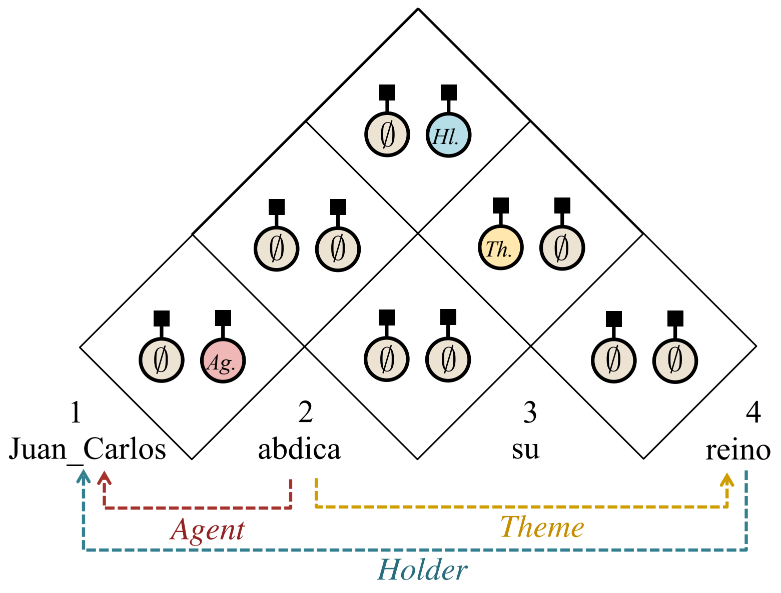


Figure 3.2: Example semantic roles for sentence and the corresponding variable assignment of the factor graph for the semantic dependency model. A parse chart is overlaid to provide a visual analogy with the parse chart in Figure 3.3. Each possible semantic edge has a single L+1-ary variable. The variable corresponding to the edge from "abdica" to "reino" has value "Theme" (abbr. Th.) corresponding to the assigned semantic role shown below the sentence. The two variables with value "Agent" (abbr. Ag.) and "Holder" (abbr. HL) indicate that "Juan_Carlos" fills two semantic roles for "abdica" and "reino" respectively. All other variables have value $\emptyset$ to indicate that they assign no semantic role.

3.2.2.1 Constrained Grammar Induction

This new grammar induction method, which we will refer to as DMV+C, induces grammar in a distantly supervised fashion by using a constrained parser in the E-step of Viterbi EM. Since the parser is part of a pipeline, we constrain it to respect the downstream SRL annotations during training. At test time, the parser is unconstrained.

Dependency-based semantic role labeling can be described as a simple structured prediction problem: the predicted structure is a labeled directed graph, where nodes correspond to words in the sentence. Each directed edge indicates that there is a predicate-argument relationship between the two words; the parent is the predicate and the child the argument. The label on the edge indicates the type of semantic relationship. Unlike syntactic dependency parsing, the graph is not required to be a tree, nor even a connected graph. Self-loops and crossing arcs are permitted.

The constrained *syntactic* DMV parser treats the semantic graph as observed, and constrains the syntactic parent to be chosen from one of the semantic parents, if there are any. See Figure 3.3 for an example. In some cases, imposing this constraint would not permit *any* projective dependency parses—in this case, we ignore the semantic constraint for that sentence. We parse with the CKY algorithm (Younger, 1967; Aho and Ullman, 1972) by utilizing a PCFG corresponding to the DMV (Cohn et al., 2010). Each chart cell allows only non-terminals compatible with the constrained sets. This can be viewed as a variation

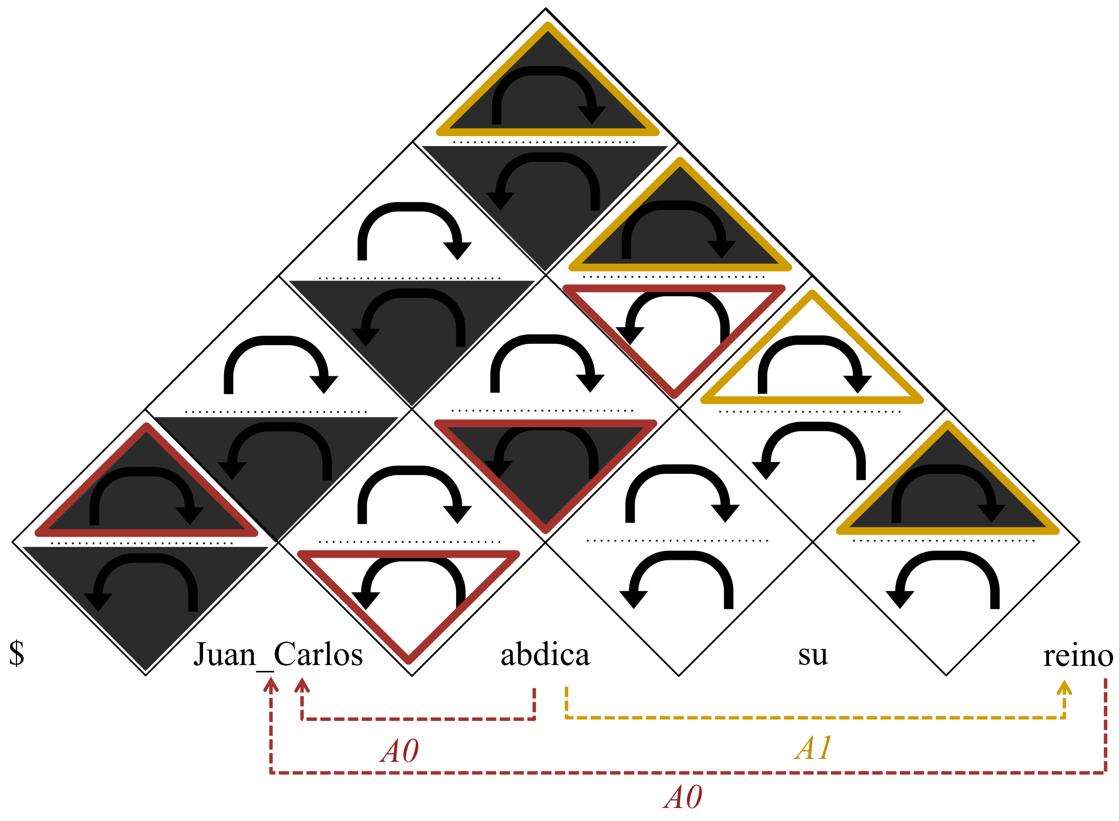


Figure 3.3: Example of a pruned parse chart for constrained grammar induction. The pruned edges are determined by the given semantic role labeling of the sentence. Each chart cell (a square) contains two possible syntactic edges: a right edge (top triangle) and a left edge (bottom triangle). For example, the yellow edge would indicate that "abdica" is the head of "reino", whereas the black edge below it indicates that "reino" heads "abdica". Only tokens filling at least one semantic role have edges to the possible parents pruned (black filled triangles). The chart cells with the word "reino" as a child correspond to the right diagonal of the chart (outlined in yellow)—all parents except for "abdica" have been pruned since it is the only available *semantic* parent. The token "Juan_Carlos" is the head of four edges (outlined in red), yet only two possible *semantic* parents—thus only the two corresponding syntactic edges are not pruned. The English gloss is "Juan_Carlos abdicates his throne", where $ indicates the special root node of the dependency parse. As usual, all edges with $ as a child are disallowed—along the left diagonal of the chart.

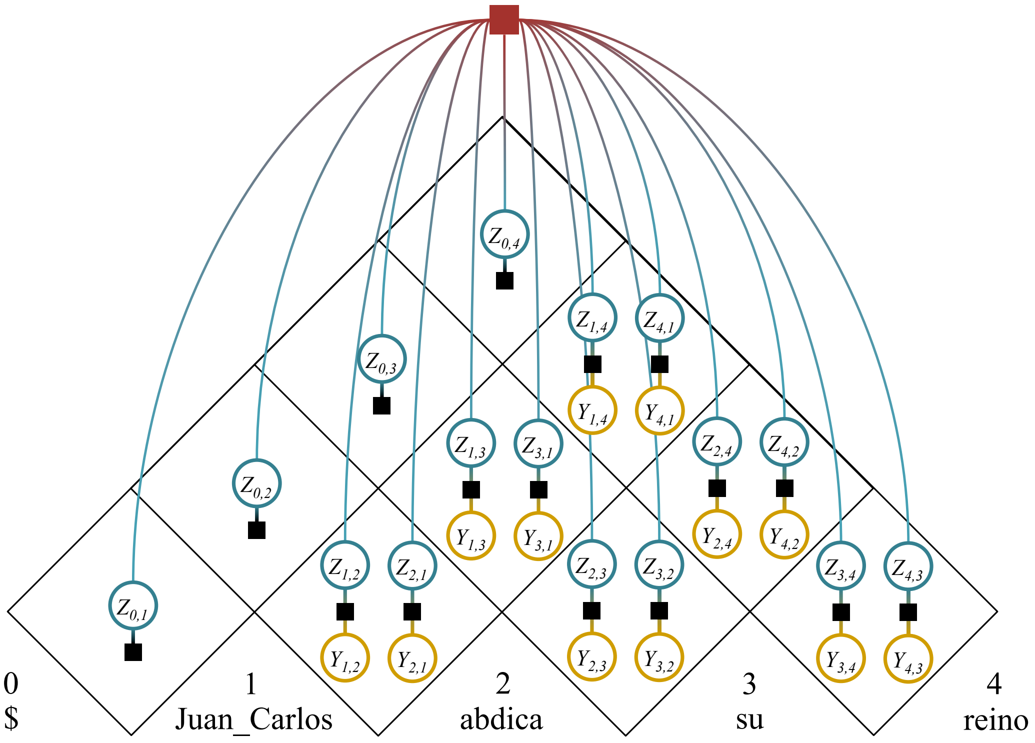


Figure 3.4: Factor graph for the joint syntactic/semantic dependency parsing model. For each of the $O(n^2)$ possible semantic edges between words i and j, there is a L+1-ary semantic role variable $Y_{i,j}$ (yellow). Each possible syntactic edge has a corresponding binary variable $Z_{i,j}$ (blue). Variable pairs are connected by factors (black). The structured PTREE factor (red) connects to the binary syntactic dependency variables and enforces that they form a projective tree. As in Figure 3.2, the special node $ is the syntactic root. For the possible syntactic edges from $i \to j$ and $j \to i$, left/right within a chart cell of this figure corresponds to top/bottom in Figure 3.3.

of Pereira and Schabes (1992).[4]

3.2.3 Joint Syntactic and Semantic Parsing Model

In Sections 3.2.1 and 3.2.2, we introduced pipeline-trained models for SRL, which used grammar induction to predict unlabeled syntactic parses. In this section, we define a simple model for joint syntactic and semantic dependency parsing. Just as in the previous pipeline methods, we use Brown clusters in place of POS tags (Section 3.2.1.1).

This model extends the CRF model in Section 3.2.1.3 to include the projective syntactic dependency parse for a sentence. This is done by including an additional n^2 binary variables that indicate whether or not a directed syntactic dependency edge exists between a pair of words in the sentence. Unlike the semantic dependencies, these syntactic variables

[4]The constrained grammar induction methods described here and our reimplementation of Spitkovsky et al. (2010a) are one of the few aspects of this thesis that is not released as part of the Pacaya framework described in Appendix A. However, we intend to release this grammar induction code separately.

must be coupled so that they produce a projective dependency parse; this requires an additional global constraint factor to ensure that this is the case (Smith and Eisner, 2008). The constraint factor touches all n^2 syntactic-edge variables, and multiplies in 1.0 if they form a projective dependency parse, and 0.0 otherwise. We couple each syntactic edge variable to its semantic edge variable with a binary factor. Figure 3.4 shows the factor graph for this joint model.

Note that our factor graph does not contain any loops, thereby permitting efficient exact marginal inference. This is an instance of structured belief propagation (cf. Section 2.3.3.4). We train our CRF models by maximizing conditional log-likelihood (cf. Section 2.3.4.1) using stochastic gradient descent with an adaptive learning rate (AdaGrad) (Duchi et al., 2011) over mini-batches of size 14 (cf. Section 2.4.2.4).

The unary and binary factors are defined with exponential family potentials. In the next section, we consider binary features of the observations (the sentence and labels from previous pipeline stages) which are conjoined with the state of the variables in the factor.

Discussion The factor graph we have presented here is almost identical to that of Naradowsky et al. (2012a). Our use of an L+1-ary semantic edge variable is in contrast to the model of Naradowsky et al. (2012a), which used a more complex set of binary variables and required a constraint factor permitting AT-MOST-ONE. Note that while these two ways of defining the variables and factors are different, they encode the same model. Practically, they differ in how we perform inference: in the Naradowsky et al. (2012a) model the factor graph has cycles and BP must be run to convergence. By contrast, our model requires only a single iteration of BP with messages passed from leaves to root and back from root to leaves (where the root is arbitrarily chosen). Both models thus permit efficient exact inference.

3.2.4 Pipeline Model with Supervised Syntax (Skyline)

Though the focus of this chapter is on low-resource SRL, we also consider a high-resource pipeline model in order to study the effectiveness of our models and features (described below) in varying levels of supervision. For this supervised pipeline, we assume that we are given either hand-annotated (i.e. gold) or automatically predicted lemmas, morphology, POS tags, and dependency parses. Given these annotations, we define features for the SRL-only model described in Section 3.2.1.3.

3.2.5 Features for CRF Models

Our feature design stems from two key ideas. First, for SRL, it has been observed that feature bigrams (the concatenation of simple features such as a predicate's POS tag and an argument's word) are important for state-of-the-art performance (Zhao et al., 2009; Björkelund et al., 2009). Second, for syntactic dependency parsing, combining Brown cluster features with word forms or POS tags yields high accuracy even with little training data (Koo et al., 2008).

We create binary indicator features for each model using feature templates. Our feature template definitions combine those used by the top performing systems in the CoNLL-2009

Property			Possible values
1	`word`(i)	word form	all word forms
2	`lc`(i)	lower case word form	all lower-case forms
3	`chpre5`(i)	5-char word form prefixes	all 5-char form prefixes
4	`cap`(i)	capitalization	*True, False*
5	`word800`(i)	top-800 word form	top-800 word forms
6	`bc`(i)	brown cluster	*000, 1100, 010110001*, ...
7	`bc5`(i)	brown cluster, length 5	length 5 prefixes of `bc`
8	`lemma`(i)	lemma	all word lemmas
9	`pos`(i)	POS tag	*NNP, CD, JJ, DT*, ...
10	`m1`(i), `m2`(i), ...	morphological features (different across languages)	Gender, Case, Number, ...
11	`deprel`(i)	dependency label	*SBJ, NMOD, LOC*, ...
12	`dir`(i)	edge direction	*Up, Down*

(a) Word and edge properties in SRL feature templates. For each property (left column) we show examples of its possible values or a brief description of those values (right column).

$\tau(i)$		$\rho(p, c)$
i, i-1, i+1	`noFarChildren`(i)	`linePath`(p, c)
`parent`(i)	`rightNearSib`(i)	`depPath`(p, c)
`allChildren`(i)	`leftNearSib`(i)	`depPath`(p, `lca`(p, c))
`rightNearChild`(i)	`firstVSupp`(i)	`depPath`(c, `lca`(p, c))
`rightFarChild`(i)	`lastVSupp`(i)	`depPath`(`lca`(p, c), \$)
`leftNearChild`(i)	`firstNSupp`(i)	
`leftFarChild`(i)	`lastNSupp`(i)	

(b) Word positions used in SRL feature templates. Based on current word position (i), positions related to current word i, possible parent, child (p, c), lowest common ancestor between parent/child (`lca`(p, c)), and syntactic root (\$).

Template	Possible values
`relative-position`	*before, after, on*
`distance`	$\mathbb{Z}^+$
`continuity`	$\mathbb{Z}^+$
`binned-distance`	$>$ 2, 5, 10, 20, 30, or 40
`geneological-relationship`	*parent, child, ancestor, descendant*
`path-grams`	*the_NN_went*

(c) Additional standalone feature templates for SRL.

Table 3.1: Feature templates for semantic role labeling

Shared Task, Zhao et al. (2009) and Björkelund et al. (2009), and from features in syntactic dependency parsing (McDonald et al., 2005; Koo et al., 2008). Though each base feature comes from prior work, some of the feature combinations we construct are novel.

As is common in CRFs, we define our features as a conjunction of an indicator function $\mathbb{I}$ for a fixed variable assignment $\tilde{\boldsymbol{y}}_\alpha$ with some *observation function* $g_{\alpha,k}(\boldsymbol{x})$ of the input sentence: $f_{\alpha,\tilde{\boldsymbol{y}}_\alpha,k}(\boldsymbol{y}_\alpha, \boldsymbol{x}) = \mathbb{I}(\tilde{\boldsymbol{y}}_\alpha = \boldsymbol{y}_\alpha) g_{\alpha,k}(\boldsymbol{x})$. In the following subsections, we describe the form of the observation functions $g_{\alpha,k}$ that we consider in terms of a feature template language.

3.2.5.1 Template Creation from Properties of Ordered Positions

The feature templates are defined using a simple template language comprised of three types of functions:

1. **Properties** The property functions map a token position in a sentence to a string. For example, given the position i, the function $\texttt{pos}(i)$ returns the part-of-speech (POS) tag for token i. We also allow these properties to be applied to a list of positions: $\texttt{pos}([4, 8, 3])$ returns a list of strings comprised of the POS tags of words 4, 8, and 3 such as [VBN, NNP, NNP].

2. **Position-modifiers** The position modifier functions come in three forms. The first form maps a single token position in a sentence to a new position: for example $\texttt{parent}(i)$ returns the syntactic head of token i. The second form maps a single token to a list of positions: $\texttt{allChildren}(i)$ returns the positions of all syntactic dependents of token i. The third form maps a pair of positions to a list of positions: $\texttt{depPath}(p, c)$ returns the positions along the dependency path of between token p and token c.

3. **Orders** We may wish to canonicalize a list of strings returned by a property applied to a list of positions. For example $\texttt{pos}(\texttt{allChildren}(i))$ returns the list of POS tags for the syntactic children of token i. The *order* function $\texttt{bag}(l)$ returns the concatenation of the sorted list of unique strings in list l. For example $\texttt{bag}([\text{VBN}, \text{NNP}, \text{NNP}]) = \text{NNP\_VBN}$. In this way, we can obtain a canonically ordered set of unique POS tags found among the syntactic children of token i using the feature template $\texttt{bag}(\texttt{pos}(\texttt{allChildren}(i)))$.

By composing these types of functions in different ways, we can obtain a wide variety of templates.[5] In the following paragraphs, we detail the properties (also listed in Table 3.1a), positions and position-modifiers (also listed in Table 3.1b), and orders. We also consider a few additional feature templates that do not fit within the property/position/order structure above: these extra features are listed in Table 3.1c and described at the end of this section. Note that this subsection only considers the base feature set. In the next subsection (Section 3.2.6), we consider how these features can be composed by concatenating multiple feature templates together—and more importantly how to select among the numerous

[5] Appendix A.2 describes the feature template language used in Pacaya which closely mirrors our feature descriptions in this section.

possibilities that result. Next, we consider how to define the observation function $g_{\alpha,k}$ from the feature templates we will define.

Observation Functions from Feature Templates Each feature template defines a mapping from a given sentence and position (or positions) to a string. From our example above, bag(pos(allChildren(i))) returns a concatenation of POS tags. We assign an integer index to each such string we observe during training. Suppose the integer k corresponds to the string "NNP_VBN", then we would define g such that $g_{\alpha,k}(\boldsymbol{x}) = 1$ if the template bag(pos(allChildren(i))) returns "NNP_VBN" and $g_{\alpha,k}(\boldsymbol{x}) = 0$ otherwise. We define g likewise for each integer k and its corresponding string / template. Note that each factor α in the factor graph may use different observation functions. In practice, we use the same observation-functions for each type of factor: e.g. all the unary factors for the role variables have the same templates and separately all the unary factors for the sense variables have the same templates.

The input positions to the feature templates are given by the variables that neighbor the factor. See Figure 3.4 for an example factor graph. Each role variable $Y_{p,c}$ is defined for a specific semantic dependency edge corresponding to a predicate token p and an argument token c. Likewise, a link variable $Z_{p,c}$ has a (p, c) pair corresponding to the positions of the parent and child for the syntactic edge. Each sense variable S_p defines a single position p corresponding to the predicate.

Property Functions The properties include word form (word), lemma (lemma), POS tag (pos), coarse (bc5) and fine-grained (bc) Brown cluster, and morphological features (m1, m2, . . .). We also include the lower-cased word (lc), the 5-character prefix if the word is more than 5 characters (chpre5), whether the word is capitalized (cap), and the word form only if it appears in the top 800 most frequent types (word800). Given a dependency path, properties include the dependency relation of each word (deprel), and the direction of the syntactic edge (dir). See Table 3.1a for example values that these properties can take.

Positions and Position-Modifier Functions Properties are extracted from word positions within the sentence. We define a set of unary operators $\tau(i)$ that generate positions for a single word i which could be either the parent or child, and a set of binary operators $\rho(p, c)$. The position operators can return either a single position or a set of positions. Word position operators are shown in Table 3.1b.

Single positions for a word i include the word to its left $(i - 1)$ and right $(i + 1)$, its syntactic parent (parent), its leftmost farthest child (leftFarChild), its rightmost nearest sibling (rightNearSib), etc. We also include the notion of verb and noun supports (Zhao et al., 2009). A *verb support* (VSupp) position for i is a position above i in the syntactic tree that has a verb POS, while a *noun support* (NSupp) has a noun POS.[6] We include two types of supports, corresponding to the support positions closest to i (first) and closest to the root (last) respectively in the syntactic tree.

[6]These features are informative only in our models that use POS tags.

Sets of positions for a word i include children and the children (`allChildren`) without the leftmost and rightmost included (`noFarChildren`). For a pair of positions (p, c), we include the list of positions in their linear path (`linePath`) from left to right. Also following previous work, we utilize four shortest dependency paths (Zhao et al., 2009) with binary operator `depPath`: from parent to child, from parent to $\texttt{lca}(p, c)$, from child to $\texttt{lca}(p, c)$, and from $\texttt{lca}(p, c)$ to root—where $\texttt{lca}(p, c)$ is the least-common-ancestor of p and c in the syntactic parse.

Order Functions Following Zhao et al. (2009), properties from a set of positions can be put together in three possible orders: as the given sequence (`seq`), as a sorted list of unique strings (`bag`), and removing all duplicated neighbored strings (`noDup`).

3.2.5.2 Additional Features

Additional templates we include are the relative position (`relative-position`) (Björkelund et al., 2009), geneological relationship (`geneological-relationship`), distance (`distance`) (Zhao et al., 2009), and binned distance (`binned-distance`) (Koo et al., 2008) between two words in the path. From Lluís et al. (2013), we use $1, 2, 3$-gram path features of words/POS tags (`path-grams`), and the number of non-consecutive token pairs in a predicate-argument path (`continuity`). Possible values these features can take are shown in Table 3.1c.

3.2.6 Feature Selection

Constructing all feature templates (herein called template unigrams) and pairs of templates (template bigrams) would yield an unwieldy number of features. We therefore include all template unigrams, but only a subset of template bigrams. These are chosen to be the top N template bigrams for a dataset and factor α according to an information gain measure (Martins et al., 2011a):

$$\mathrm{IG}_{\alpha,m} = \sum_{f \in T_m} \sum_{\boldsymbol{y}_\alpha} p(f, \boldsymbol{y}_\alpha) \log_2 \frac{p(f, \boldsymbol{y}_\alpha)}{p(f)p(\boldsymbol{y}_\alpha)} \tag{3.1}$$

where T_m is the mth feature template, f is a particular instantiation of that template, and $\boldsymbol{y}_\alpha$ is an assignment to the variables in factor α. The probabilities are empirical estimates computed from the training data. This is simply the mutual information of the feature template instantiation with the variable assignment.

This filtering approach was treated as a simple baseline in Martins et al. (2011a) to contrast with increasingly popular gradient-based regularization approaches. Unlike the gradient-based approaches, this filtering approach easily scales to many features since we can decompose the memory usage over feature templates.

As an additional speedup, we reduce the dimensionality of our feature space to 1 million for each clique using a common trick referred to as *feature hashing* (Weinberger et al., 2009): we map each feature instantiation to an integer using a hash function[7] modulo the desired dimensionality.

Example feature templates Table 3.2 shows the bigram feature templates that were selected according to the information gain criterion on the English CoNLL-2009 data. The selected templates highlight a key shortcoming of the approach: because the information gain computation is independent for each feature, there is nothing to encourage diversity among the selected features. As such, the top features are highly redundant. For example, the first four templates conjoin `word800`(c) with either the word, lower-cased word, 5-character prefix of the word, or its lemma.

3.3 Related Work

Our work builds upon research in both semantic role labeling and unsupervised grammar induction (Klein and Manning, 2004; Spitkovsky et al., 2010a). Previous related approaches to semantic role labeling include joint classification of semantic arguments (Toutanova et al., 2005; Johansson and Nugues, 2008), latent syntax induction (Boxwell et al., 2011; Naradowsky et al., 2012a), and feature engineering for SRL (Zhao et al., 2009; Björkelund et al., 2009).

High-resource SRL As discussed in the introduction, semantic role labeling is traditionally approached by first identifying syntactic features of the sentence and then predicting predicates and their arguments. These often use a pipeline of classifiers for predicate disambiguation, argument identification, and argument classification (Gildea and Jurafsky, 2000; Gildea and Jurafsky, 2002; Surdeanu et al., 2008). Such pipeline approaches rely heavily on the accuracy of the syntactic parser (Gildea and Palmer, 2002; Punyakanok et al., 2005). This decomposition prohibits the parser from utilizing the labels from the end task.

Toutanova et al. (2005) introduced one of the first joint approaches for SRL and demonstrated that a model that scores the full predicate-argument structure of a parse tree could lead to significant error reduction over independent classifiers for each predicate-argument relation.

Johansson and Nugues (2008) and Lluís et al. (2013) extend this idea by coupling predictions of a dependency parser with predictions from a semantic role labeler. In the model from Johansson and Nugues (2008), the outputs from an SRL pipeline are reranked based on the full predicate-argument structure that they form. The candidate set of syntactic-semantic structures is reranked using the probability of the syntactic tree and semantic structure. Lluís et al. (2013) use a joint arc-factored model that predicts full syntactic paths along with predicate-argument structures via dual decomposition.

Low-resource SRL Boxwell et al. (2011) and Naradowsky et al. (2012a) observe that syntax may be treated as latent when a treebank is not available. Boxwell et al. (2011) describe a method for training a semantic role labeler by extracting features from a packed

[7]To reduce hash collisions, we use MurmurHash v3 `https://code.google.com/p/smhasher`.

Bigram Template	$\text{IG}_{a,m}$
`word800`(c) + `word`(p)	2.841077
`word800`(c) + `lc`(p)	2.840362
`word800`(c) + `chpre5`(p)	2.822854
`word800`(c) + `lemma`(p)	2.815270
`word`(c) + `word`(p)	2.738279
`lemma`(c) + `word`(p)	2.735499
`word`(c) + `lc`(p)	2.735469
`lc`(c) + `lc`(p)	2.735363
`chpre5`(c) + `lc`(p)	2.733696
`chpre5`(c) + `word`(p)	2.733115
`lemma`(c) + `lc`(p)	2.732447
`lc`(c) + `word`(p)	2.731534
`bc`(c) + `lc`(p)	2.726672
`bc`(c) + `word`(p)	2.725611
`word`(c) + `chpre5`(p)	2.725222
`lemma`(c) + `chpre5`(p)	2.720818
`lc`(c) + `chpre5`(p)	2.719960
`word`(c) + `lemma`(p)	2.719056
`chpre5`(c) + `chpre5`(p)	2.716669
`chpre5`(c) + `lemma`(p)	2.713464
`lc`(c) + `lemma`(p)	2.712567
`lemma`(c) + `lemma`(p)	2.711402
`bc`(c) + `chpre5`(p)	2.709945
`word800`(c) + `word800`(p)	2.707732
`seq(pos(linePath`(p,c)`))` + `word`(p)	2.704583
`seq(pos(linePath`(p,c)`))` + `lemma`(p)	2.702469
`seq(pos(linePath`(p,c)`))` + `lc`(p)	2.701657
`bc`(c) + `lemma`(p)	2.695978
`word800`(c) + `bc`(p)	2.695099
`seq(pos(linePath`(p,c)`))` + `chpre5`(p)	2.689965
`seq(deprel(linePath`(p,c)`))` + `word`(p)	2.685279
`seq(deprel(linePath`(p,c)`))` + `lc`(p)	2.683995

Table 3.2: Top 32 bigram feature templates ranked by information gain ($\text{IG}_{a,m}$) on semantic argument classification the first 1000 sentences of the English CoNLL-2009 data. + indicates the conjoining of two templates. See Section 3.2.5 for a description of the feature template language used here. p is the position of the predicate and c the position of the argument.

CCG parse chart, where the parse weights are given by a simple ruleset. Naradowsky et al. (2012a) marginalize over latent syntactic dependency parses.

Both Boxwell et al. (2011) and Naradowsky et al. (2012a) suggest methods for SRL without supervised syntax, however, their features come largely from supervised resources. Even in their lowest resource setting, Boxwell et al. (2011) require an oracle CCG tag dictionary extracted from a treebank. Naradowsky et al. (2012a) limit their exploration to a small set of basic features, and included high-resource supervision in the form of lemmas, POS tags, and morphology available from the CoNLL 2009 data.

There has not yet been a comparison of techniques for SRL that do not rely on a syntactic treebank, and no exploration of probabilistic models for unsupervised grammar induction within an SRL pipeline that we have been able to find.

Grammar Induction Related work for the unsupervised learning of dependency structures separately from semantic roles primarily comes from Klein and Manning (2004), who introduced the Dependency Model with Valence (DMV). This is a robust generative model that uses a head-outward process over word classes, where heads generate arguments.

Grammar induction work has further demonstrated that distant supervision in the form of ACE-style relations (Naseem and Barzilay, 2011) or HTML markup (Spitkovsky et al., 2010b) can lead to considerable gains. Recent work in fully unsupervised dependency parsing has supplanted these methods with even higher accuracies (Spitkovsky et al., 2013) by arranging optimizers into networks that suggest informed restarts based on previously identified local optima. We do not reimplement these approaches within the SRL pipeline here, but provide comparison of these methods against our grammar induction approach in isolation in Section 3.5.4.

Feature Templates for SRL In both pipeline and joint models, we use features adapted from state-of-the-art approaches to SRL (Section 3.2.5). This includes Zhao et al. (2009) features, who use feature templates from combinations of word properties, syntactic positions including head and children, and semantic properties; and features from Björkelund et al. (2009), who utilize features on syntactic siblings and the dependency path concatenated with the direction of each edge.

3.4 Experimental Setup

3.4.1 Data

The CoNLL-2009 Shared Task (Hajič et al., 2009) dataset contains POS tags, lemmas, morphological features, syntactic dependencies, predicate senses, and semantic roles annotations for 7 languages: Catalan, Chinese, Czech, English, German, Japanese,[8] Spanish. The CoNLL-2005 and -2008 Shared Task datasets provide English SRL annotation, and for cross-dataset comparability we consider only verbal predicates. To compare with prior

[8]We do not report results on Japanese as that data was only made freely available to researchers that competed in CoNLL 2009.

approaches that use semantic supervision for grammar induction, we utilize Section 23 of the WSJ portion of the Penn Treebank (Marcus et al., 1993).

The CoNLL-2005, CoNLL-2008, and English CoNLL-2009 datasets share the same underlying sentences from the Penn Treebank. However, the exact set of sentences included in the respective shared tasks differ slightly. The argument annotations in CoNLL-2005 are over spans, whereas the heads of arguments are annotated in CoNLL-2008 and -2009. These details are further described in Section 3.5.2.

The standard evaluation for CoNLL-2009 includes sense prediction as well as argument identification and classification. While we view sense prediction as somewhat tangential to our discussion of SRL, we include it in some of the evaluation metrics in order to compare to prior work.

3.4.2 Feature Template Sets

We consider two sets of features based on the information gain feature selection criterion. Our primary feature set, $\mathbf{IG}_C$, consists of 127 template unigrams. Note that this is not every possible feature template describable by our feature template language, but rather a subset that emphasizes coarse word properties (i.e., properties 7, 9, and 11 in Table 3.1a). We also explore the 31 template unigrams[9] described by Björkelund et al. (2009), here referred to as $\mathbf{IG}_B$. Each of IG_C and IG_B also include 32 template bigrams selected by information gain on 1000 sentences—we select a different set of template bigrams for each dataset.

We compare against the language-specific feature sets detailed in the literature on high-resource top-performing SRL systems: From Björkelund et al. (2009), these are feature sets for German, English, Spanish and Chinese, obtained by weeks of forward selection ($\mathbf{B}_{de,en,es,zh}$); and from Zhao et al. (2009), these are features for Catalan $\mathbf{Z}_{ca}$.[10]

3.5 Results

We are interested in the effects of varied supervision using pipeline and joint training for SRL. To compare to prior work (i.e., submissions to the CoNLL-2009 Shared Task), we also consider the joint task of semantic role labeling *and* predicate sense disambiguation. Our experiments are subtractive, beginning with all supervision available and then successively removing (a) dependency syntax, (b) morphological features, (c) POS tags, and (d) lemmas. Dependency syntax is the most expensive and difficult to obtain of these various forms of supervision. We explore the importance of both the labels and structure, and what quantity of supervision is useful.

[9] Because we do not include a binary factor between predicate sense and semantic role, we do not include sense as a feature for argument prediction.

[10] This covers all CoNLL languages but Czech, where feature sets were not made publicly available in either work. In Czech, we disallowed template bigrams involving `path-grams`.

SRL Approach	Feature Set	Avg. F1
Pipeline	IG_C	**84.98**
Pipeline	IG_B	84.74
Naradowsky et al. (2012a)		72.73

Table 3.3: Test F1 of supervised SRL and sense disambiguation with *gold* (oracle) syntax averaged over the CoNLL-2009 languages. See Table 3.6(a) for per-language results.

3.5.1 CoNLL-2009: High-resource SRL

We first compare our models trained as a pipeline, using all available supervision (syntax, morphology, POS tags, lemmas) from the CoNLL-2009 data.

Gold Syntax Table 3.3 shows the results of our model with gold syntax and a richer feature set than that of Naradowsky et al. (2012a), which only looked at whether a syntactic dependency edge was present. Table 3.6(a) contains the full per-language results for the averages shown in Table 3.3 as well as additional feature template settings (Z_{ca}, $\text{B}_{de,en,es,zh}$). Recall an important advantage of the pipeline trained model: the features can consider any part of the syntax (e.g., arbitrary subtrees), whereas the joint model is limited to those features over which it can efficiently marginalize (e.g., short dependency paths). This holds true even in the pipeline setting where no syntactic supervision is available. The contrast of our pipeline with rich features on gold syntax (84.98 Avg. F1) vs. Naradowsky et al. (2012a) which uses a limited feature set on gold syntax (72.73 Avg. F1) demonstrates the importance of rich features in the high-resource setting: without such features performance suffers (a 12.25 drop in Avg. F1). Including such features in a joint model requires approximate inference such as BP or dual decomposition (cf. Lluís et al. (2013)). In this chapter, we have restricted our attention to joint models for which exact inference is possible (i.e. without rich features), yet we will observe that even with a limited feature set, the joint model can outperform a feature-rich model in the *low*-resource setting.

Supervised Syntax Table 3.4 uses predicted (not gold) syntax and contrasts our high-resource results for the task of SRL and sense disambiguation with the top systems in the CoNLL-2009 Shared Task, giving further insight into the performance of the simple information gain feature selection technique. With supervised syntax, the simple information gain feature selection technique (Section 3.2.6) applied to a large set of 127 template unigrams IG_C performs admirably (78.03 Avg. F1), but still lags behind the state-of-the-art from Björkelund et al. (2009) (81.55 Avg. F1). Using only a reduced set of 32 template unigrams IG_B for feature selection, we see a drop in performance (75.68 Avg. F1).

Table 3.6(b) presents the per-language results corresponding to the averages in Table 3.4 as well as results with language-specific feature sets from prior work (Z_{ca}, $\text{B}_{de,en,es,zh}$). Here we can contrast two feature selection techniques: The original Björkelund features ($\text{B}_{de,en,es,zh}$) were selected by a computationally intensive search through possible template bigrams: each bigram was added to the model and kept only if the F1 score of the high-resource model improved on development data. Our feature set IG_B began with the same set of template unigrams, but selected the template bigrams by a (comparatively simple)

SRL Approach	Feature Set	Avg. F1
Björkelund et al. (2009)		**81.55**
Zhao et al. (2009)		80.85
Pipeline	IG_C	78.03
Pipeline	IG_B	75.68

Table 3.4: Test F1 of supervised SRL and sense disambiguation with *supervised* syntax, averaged over CoNLL-2009 languages. See Table 3.6(b) for per-language results.

information gain criterion. Both approaches select a different set of features for each language. For each of German, English, Spanish, and Chinese, $\text{B}_{de,en,es,zh}$ yields higher F1 than our information gain set IG_B. This suggests that further work on feature selection may improve the results. For an additional comparison, we look ahead to the low-resource results in Table 3.6(c) that will be discussed in Section 3.5.2: we find that IG_B obtain *higher* F1 than the original Björkelund feature sets $\text{B}_{de,en,es,zh}$ in the low-resource pipeline setting with constrained grammar induction (DMV+C)—this is the opposite of high-resource result. This suggests that the $\text{B}_{de,en,es,zh}$ are over-tuned to the high-resource setting on which they were selected, and as a result perform poorly in the low-resource setting.

3.5.2 CoNLL-2009: Low-Resource SRL

In this section, we contrast our three approaches (Sections 3.2.1, 3.2.2, 3.2.3) to handling the case where we have supervised data for semantic roles, but have no syntactic training data available. Then we consider an even lower-resource setting in which we subtract out other forms of syntactic supervision: morphology, lemmas, and POS tags. The key takeaway is that a pipeline permits rich features of previous stages, but doesn't permit errors to propagate between stages. By contrast, a joint model might not be able to incorporate the same rich features efficiently, but it allows confidence in one part of the model (e.g. syntax) to influence another (e.g. semantics).

Latent Syntax Table 3.5 includes results for our low-resource approaches and Naradowsky et al. (2012a) on predicting semantic roles as well as sense. See Table 3.6(c) for per-language results and additional features not shown in Table 3.5. In the low-resource setting of the CoNLL-2009 Shared task without syntactic supervision, our joint model (Joint) with marginalized syntax obtains state-of-the-art results with features IG_C described in Section 3.4.2. This model outperforms prior work (Naradowsky et al., 2012a) and our pipeline model (Pipeline) with constrained (DMV+C) and unconstrained grammar induction (DMV) trained on brown clusters (bc).

These results begin to answer a key research question in this work: The joint models outperform the pipeline models in the low-resource setting. This holds even when using the same feature selection process. Further, the best-performing low-resource features found in this work are those based on coarse feature templates and selected by information gain. Templates for these features generalize well to the high-resource setting. However, analysis of the induced grammars in the pipeline setting suggests that the book is not closed on the

SRL Approach	**Feature Set**	**Dep. Parser**	**Avg. F1**
Joint	IG$_C$	Marginalized	**72.48**
Joint	IG$_B$	Marginalized	72.40
Naradowsky et al. (2012a)		Marginalized	71.27
Pipeline	IG$_C$	DMV+C (bc)	70.08
Pipeline	IG$_C$	DMV (bc)	69.26
Pipeline	IG$_B$	DMV (bc)	66.81
Pipeline	IG$_B$	DMV+C (bc)	65.61

Table 3.5: Test F1 of supervised SRL and sense disambiguation with *no supervision* for syntax, averaged over CoNLL-2009 languages. See Table 3.6(c) for per-language results.

issue. We return to this in Section 3.5.4.

Training and decoding times for the pipeline and joint methods are similar as computation time tends to be dominated by feature extraction.

Subtractive Study In our subsequent experiments, we study the effectiveness of our models as the available supervision is decreased. We incrementally remove dependency syntax, morphological features, POS tags, then lemmas. For these experiments, we utilize the coarse-grained feature set (IG$_C$), which includes Brown clusters.

Across languages, we find the largest drop in F1 when we remove POS tags; and we find a gain in F1 when we remove lemmas. This indicates that lemmas, which are a high-resource annotation, may not provide a significant benefit for this task. The effect of removing morphological features is different across languages, with little change in performance for Catalan and Spanish, but a drop in performance for German. This may reflect a difference between the languages, or may reflect the difference between the annotation of the languages: both the Catalan and Spanish data originated from the Ancora project,[11] while the German data came from another source.

Figure 3.5 contains the learning curve for SRL supervision in our lowest resource setting for two example languages, Catalan and German. This shows how F1 of SRL changes as we adjust the number of training examples. We find that the joint training approach to grammar induction yields consistently higher SRL performance than its distantly supervised counterpart.

3.5.3 CoNLL-2008, -2005 without a Treebank

In this section, we return to the "no syntax" setting of the previous section. We do so in order to contrast our dependency-based SRL models with that of a state-of-the-art span-based SRL model. This provides the first such comparison in the low-resource setting.

We contrast our approach with that of Boxwell et al. (2011), who evaluate on semantic argument identification and classification in isolation—that is, they do not include predicate sense disambiguation, as is done in the CoNLL-2009 Shared Task benchmark. They report results on Prop-CCGbank (Boxwell and White, 2008), which uses the same training/testing

[11]http://clic.ub.edu/corpus/ancora

	SRL Approach	**Feature Set**	**Dep. Parser**	**Avg.**	**ca**	**cs**	**de**	**en**	**es**	**zh**
(a)	Pipeline	IG_C	Gold	**84.98**	84.97	**87.65**	79.14	**86.54**	84.22	**87.35**
	Pipeline	Z_{ca}	Gold	*86.49	86.49	—	—	—	—	—
	Pipeline	IG_B	Gold	84.74	**85.15**	86.64	**79.50**	85.77	**84.40**	86.95
	Pipeline	$\text{B}_{de,en,es,zh}$	Gold	*83.80	—	—	77.06	85.62	85.49	87.03
	Naradowsky et al. (2012a)		Gold	72.73	69.59	74.84	66.49	78.55	68.93	77.97
(b)	Björkelund et al. (2009)		Supervised	**81.55**	80.01	**85.41**	**79.71**	**85.63**	79.91	**78.60**
	Zhao et al. (2009)		Supervised	80.85	**80.32**	85.19	75.99	85.44	**80.46**	77.72
	Pipeline	IG_C	Supervised	78.03	76.24	83.34	74.19	81.96	76.12	76.35
	Pipeline	Z_{ca}	Supervised	*77.62	77.62	—	—	—	—	—
	Pipeline	$\text{B}_{de,en,es,zh}$	Supervised	*76.49	—	—	72.17	81.15	76.65	75.99
	Pipeline	IG_B	Supervised	75.68	74.59	81.61	69.08	78.86	74.51	75.44
(c)	Joint	IG_C	Marginalized	**72.48**	71.35	**81.03**	65.15	**76.16**	71.03	70.14
	Joint	IG_B	Marginalized	72.40	**71.55**	80.04	64.80	75.57	**71.21**	71.21
	Naradowsky et al. (2012a)		Marginalized	71.27	67.99	73.16	**67.26**	76.12	66.74	**76.32**
	Joint	Z_{ca}	Marginalized	*70.98	70.98	—	—	—	—	—
	Pipeline	IG_C	DMV+C (bc)	70.08	68.21	79.63	62.25	73.81	68.73	67.86
	Pipeline	Z_{ca}	DMV+C (bc)	*69.67	69.67	—	—	—	—	—
	Pipeline	IG_C	DMV (bc)	69.26	68.04	79.58	58.47	74.78	68.36	66.35
	Pipeline	IG_B	DMV (bc)	66.81	63.31	77.38	59.91	72.02	65.96	62.28
	Joint	$\text{B}_{de,en,es,zh}$	Marginalized	*65.63	—	—	60.05	71.06	65.34	66.07
	Pipeline	IG_B	DMV+C (bc)	65.61	61.89	77.48	58.97	69.11	63.31	62.92
	Pipeline	$\text{B}_{de,en,es,zh}$	DMV+C (bc)	*63.06	—	—	57.75	68.32	63.70	62.45

Table 3.6: Test F1 for SRL and sense disambiguation on CoNLL'09 in high-resource and low-resource settings: we study (a) gold syntax, (b) supervised syntax, and (c) unsupervised syntax. Results are ranked by F1 with bold numbers indicating the best F1 for a language and level of supervision. *Indicates partial averages for the language-specific feature sets (Z_{ca} and $\text{B}_{de,en,es,zh}$), for which we show results only on the languages for which the sets were publicly available.

Rem	**#FT**	**ca**	**de**	**es**
–	127+32	74.46	72.62	74.23
Dep	40+32	67.43	64.24	67.18
Mor	30+32	67.84	59.78	66.94
POS	23+32	64.40	54.68	62.71
Lem	21+32	64.85	54.89	63.80

Table 3.7: Subtractive experiments. Each row contains the F1 for SRL only (without sense disambiguation) where the supervision type of that row and all above it have been removed. Removed supervision types (Rem) are: syntactic dependencies (*Dep*), morphology (*Mor*), POS tags (*POS*), and lemmas (*Lem*). #FT indicates the number of feature templates used (unigrams+bigrams).

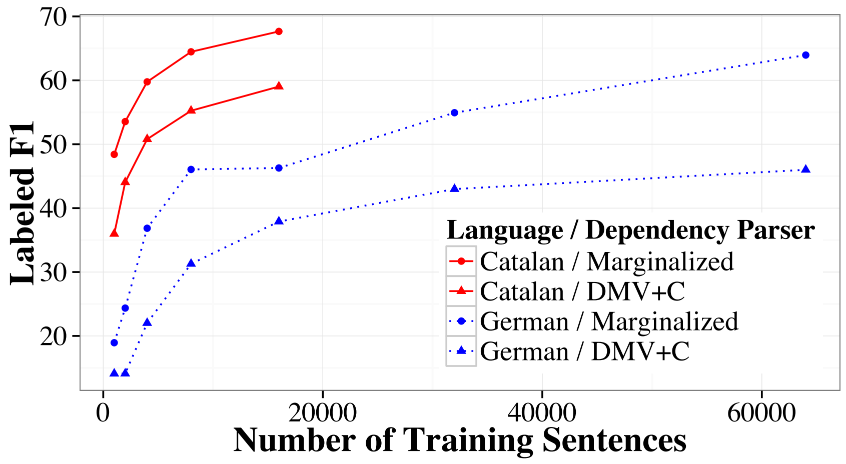


Figure 3.5: Learning curve for semantic dependency supervision in Catalan and German. F1 of SRL only (without sense disambiguation) shown as the number of training sentences is increased.

splits as the CoNLL-2005 Shared Task. Their results are therefore loosely[12] comparable to results on the CoNLL-2005 dataset.

There is an additional complication in comparing SRL approaches directly: The CoNLL-2005 dataset defines arguments as *spans* instead of heads, which runs counter to our head-based syntactic representation. This creates a mismatched train/test scenario: we must train our model to predict argument *heads*, but then test on our model's ability to predict argument *spans*.[13] We therefore train our models on the CoNLL-2008 argument heads,[14]

[12]The comparison is imperfect for two reasons: first, the CCGBank contains only 99.44% of the original PTB sentences (Hockenmaier and Steedman, 2007); second, because PropBank was annotated over CFGs, after converting to CCG only 99.977% of the argument spans were exact matches (Boxwell and White, 2008). However, this comparison was adopted by Boxwell et al. (2011), so we use it here.

[13]We were unable to obtain the system output of Boxwell et al. (2011) in order to convert their spans to dependencies and evaluate the other mismatched train/test setting.

[14]CoNLL-2005, -2008, and -2009 were derived from PropBank and share the same source text; -2008 and

train \ test		*2008 heads*	*2005 spans*	*2005 spans (oracle tree)*
☑ PRY'08	*2005 spans*	84.32	79.44	
☐ B'11 (tdc)		—	71.5	
☐ B'11 (td)		—	65.0	
☑ JN'08	*2008 heads*	85.93	79.90	
☐ Joint, IG$_C$		72.9	35.0	72.0
☐ Joint, IG$_B$		67.3	37.8	67.1

Table 3.8: F1 for SRL approaches (without sense disambiguation) in matched and mismatched train/test settings for CoNLL 2005 span and 2008 head supervision. We contrast low-resource (☐) and high-resource settings (☑), where the latter uses a treebank. See Section 3.5.2 for caveats to this comparison.

and post-process and convert from heads to spans using the conversion algorithm available from Johansson and Nugues (2008).[15] The heads are either from an MBR tree or an oracle tree. This gives Boxwell et al. (2011) the advantage, since our syntactic dependency parses are optimized to pick out semantic argument heads, not spans.

Table 3.8 presents our results. Boxwell et al. (2011) (B'11) uses additional supervision in the form of a CCG tag dictionary derived from supervised data with (tdc) and without (tc) a cutoff. Our model does very poorly on the '05 span-based evaluation because the constituent bracketing of the marginalized trees are inaccurate. This is elucidated by instead evaluating on the oracle spans, where our F1 scores are higher than Boxwell et al. (2011). We also contrast with relevant high-resource methods with span/head conversions from Johansson and Nugues (2008): Punyakanok et al. (2008) (PRY'08) and Johansson and Nugues (2008) (JN'08).

3.5.4 Analysis of Grammar Induction

Table 3.9 shows grammar induction accuracy in low-resource settings. We find that the gap between the supervised parser and the unsupervised methods is quite large, despite the reasonable accuracy both methods achieve for the SRL end task. This suggests that refining the low-resource grammar induction methods may lead to gains in SRL.

The marginalized grammars best the DMV grammar induction method; however, this difference is less pronounced when the DMV is constrained using SRL labels as distant supervision. This could indicate that a better model for grammar induction would result in better performance for SRL. We therefore turn to an analysis of other approaches to grammar induction in Table 3.10, evaluated on the Penn Treebank. We contrast with methods using distant supervision (Naseem and Barzilay, 2011; Spitkovsky et al., 2010b) and fully unsupervised dependency parsing (Spitkovsky et al., 2013). Following prior work, we

-2009 use argument heads.

[15]Specifically, we use their Algorithm 2, which produces the span dominated by each argument, with special handling of the case when the argument head dominates that of the predicate. Also following Johansson and Nugues (2008), we recover the '05 sentences missing from the '08 evaluation set.

Dependency Parser	**Avg.**	**ca**	**cs**	**de**	**en**	**es**	**zh**
Supervised*	87.1	89.4	85.3	89.6	88.4	89.2	80.7
DMV (pos)	30.2	45.3	22.7	20.9	32.9	41.9	17.2
DMV (bc)	22.1	18.8	32.8	19.6	22.4	20.5	18.6
DMV+C (pos)	37.5	50.2	34.9	21.5	36.9	49.8	32.0
DMV+C (bc)	40.2	46.3	37.5	28.7	40.6	50.4	37.5
Marginal, IG$_C$	43.8	50.3	45.8	27.2	44.2	46.3	48.5
Marginal, IG$_B$	50.2	52.4	43.4	41.3	52.6	55.2	56.2

Table 3.9: Unlabeled directed dependency accuracy on CoNLL'09 test set in low-resource settings. DMV models are trained on either POS tags (pos) or Brown clusters (bc). *Indicates the supervised parser outputs provided by the CoNLL'09 Shared Task.

	WSJ$^\infty$	**Distant Supervision**
SAJM'10	44.8	none
SAJ'13	**64.4**	none
SJA'10	50.4	HTML
NB'11	59.4	ACE05
DMV (bc)	24.8	none
DMV+C (bc)	44.8	SRL
Marginalized, IG$_C$	48.8	SRL
Marginalized, IG$_B$	58.9	SRL

Table 3.10: Comparison of grammar induction approaches on the Penn Treebank. We contrast the DMV trained with Viterbi EM+uniform initialization (DMV), our constrained DMV (DMV+C), and our model's MBR decoding of latent syntax (Marginalized) with other recent work: Spitkovsky et al. (2010a) (SAJM'10), Spitkovsky et al. (2010b) (SJA'10), Naseem and Barzilay (2011) (NB'11), and the CS model of Spitkovsky et al. (2013) (SAJ'13).

exclude punctuation from evaluation and convert the constituency trees to dependencies.[16]

The approach from Spitkovsky et al. (2013) (SAJ'13) outperforms all other approaches, including our marginalized settings. We therefore may be able to achieve further gains in the pipeline model by considering better models of latent syntax, or better search techniques that break out of local optima. Similarly, improving the nonconvex optimization of our latent-variable CRF (Marginalized) may offer further gains.

3.6 Summary

We have compared various approaches for low-resource semantic role labeling at the state-of-the-art level. We find that we can outperform prior work in the low-resource setting by

[16]Naseem and Barzilay (2011) and our results use the Penn converter (Pierre and Heiki-Jaan, 2007). Spitkovsky et al. (2010; 2013) use Collins (1999) head percolation rules.

coupling the selection of feature templates based on information gain with a joint model that marginalizes over latent syntax.

We utilize unlabeled data in both generative and discriminative models for dependency syntax and in generative word clustering. Our discriminative joint models treat latent syntax as a structured-feature to be optimized for the end-task of SRL, while our other grammar induction techniques optimize for unlabeled data likelihood—optionally with distant supervision. We observe that careful use of these unlabeled data resources can improve performance on the end task.

Our subtractive experiments suggest that lemma annotations, a high-resource annotation, may not provide a large benefit for SRL. Our grammar induction analysis indicates that relatively low accuracy can still result in reasonable SRL predictions; still, the models do not perform at the level of those that use supervised syntax. In future work, we hope to explore how well the pipeline models in particular improve when we apply higher accuracy unsupervised grammar induction techniques.

Chapter 4

Neural and Log-linear Factors

Over a decade of research has been spent carefully crafting features by hand for the task of relation extraction (Zelenko et al., 2003; Culotta and Sorensen, 2004; Bunescu and Mooney, 2005; Jiang and Zhai, 2007; Sun et al., 2011; Plank and Moschitti, 2013; Nguyen and Grishman, 2014). Yet, there has been much recent effort in attempting to show that these sorts of features can be learned automatically with neural networks. One of the conveniences of the proposed framework of this thesis is that these neural networks can easily be incorporated. However, for areas in which decades of research have been spent carefully crafting features by hand, does the application-blind machinery of neural networks still have something to offer?

In this chapter,[1] we pose this question and study it experimentally. We augment a baseline relation extraction system consisting of handcrafted features with a state-of-the-art neural network architecture—this *hybrid* model is the focus of this chapter. Our goal is to demonstrate the complementarity of the two submodels. There have been increasingly many results that suggest handcrafted features are complementary to those learned by (current) neural networks; see for example Socher et al. (2012). Similar results in relation extraction (Hashimoto et al., 2015; Liu et al., 2015) and constituency parsing (Durrett and Klein, 2015) appeared while this thesis was in preparation. However, we believe that ours is a particularly salient testing ground for the question at hand since we start with a neural network which *itself* is infused with carefully constructed real-world knowledge of the domain.

This chapter serves a secondary goal: to demonstrate the ease with which neural networks fit into our framework. Our full hybrid model provides for one of the simplest examples of training in our framework—in the case where inference (as in the previous chapter) makes no approximations. We reserve the case of approximate inference for the final two chapters.

4.1 Introduction

Two common NLP feature types are lexical properties of words and unlexicalized linguistic/structural interactions between words. Prior work on relation extraction has extensively

[1]A previous version of this work was presented in Gormley et al. (2015b).

Class	Sentence Snippet
(a) ART(M_1,M_2)	$[_{M_1}$ *A man* $]$ *driving what appeared to be* $[_{M_2}$ *a taxicab* $]$
(b) PART-WHOLE(M_1,M_2)	*direction of* $[_{M_1}$ *the southern suburbs* $]$ *of* $[_{M_2}$ *Baghdad* $]$
(c) PHYSICAL(M_2,M_1)	*in* $[_{M_1}$ *the united states* $]$, $[_{M_2}$ *284 people* $]$ *died*

Table 4.1: Examples from ACE 2005. In (a), the word "driving" is a strong indicator of the relation *ART* between M_1 and M_2. A feature that depends on the embedding for this context word could generalize to other lexical indicators of the same relation (e.g. "operating") that don't appear with *ART* during training. But lexical information alone is insufficient; relation extraction requires the identification of lexical roles: *where* a word appears structurally in the sentence. In (b), the word "of" between "suburbs" and "Baghdad" suggests that the first entity is part of the second, yet the earlier occurrence after "direction" is of no significance to the relation. Even finer information can be expressed by a word's role on the dependency path between entities. In (c), we can distinguish the word "died" from other irrelevant words that don't appear between the entities.

studied how to design such features by combining *discrete* lexical properties (e.g. the identity of a word, its lemma, its morphological features) with aspects of a word's linguistic context (e.g. whether it lies between two entities or on a dependency path between them). While these help learning, they make generalization to unseen words difficult. An alternative approach to capturing lexical information relies on *continuous* word embeddings.[2] Embedding features have improved many tasks, including NER, chunking, dependency parsing, semantic role labeling, and relation extraction (Miller et al., 2004; Turian et al., 2010; Koo et al., 2008; Roth and Woodsend, 2014; Sun et al., 2011; Plank and Moschitti, 2013; Nguyen and Grishman, 2014). Embeddings can capture lexical information, but alone they are insufficient: in state-of-the-art systems, they are used alongside features of the broader linguistic context.

In this chapter, we introduce a hybrid log-linear and neural network model for relation extraction, a task in which contextual feature construction plays a major role in generalizing to unseen data. Our baseline log-linear model directly uses handcrafted *lexicalized* features. The compositional model combines *un*lexicalized linguistic context and word embeddings. We also show that this hybrid model is a (very simple) factor graph comprised of a single variable and two factors: one factor has the standard exponential family form (i.e. the log-linear model) and the other is a less-traditional factor (i.e. the neural network model).

The compositional model is called the Feature-rich Compositional Embedding Model (FCM) and was introduced in Gormley et al. (2015b). FCM allows for the composition of embeddings with arbitrary linguistic structure, as expressed by hand crafted features. In the following sections, we describe the model starting with a precise construction of compositional embeddings using word embeddings in conjunction with unlexicalized features. Various feature sets used in prior work (Turian et al., 2010; Nguyen and Grishman, 2014;

[2]Such embeddings have a long history in NLP, including term-document frequency matrices and their low-dimensional counterparts obtained by linear algebra tools (LSA, PCA, CCA, NNMF), Brown clusters, random projections and vector space models. Recently, neural networks / deep learning have provided several popular methods for obtaining such embeddings.

Hermann et al., 2014; Roth and Woodsend, 2014) are captured as special cases of this construction. Adding these compositional embeddings directly to a standard log-linear model yields a special case of the full FCM model. Treating the word embeddings as parameters gives rise to the powerful, efficient, and easy-to-implement *log-bilinear model*. The model capitalizes on arbitrary types of linguistic annotations by better utilizing features associated with substructures of those annotations, including global information. Features are chosen to promote different properties and to distinguish different functions of the input words.

Our full *hybrid* model involves four stages. First, it decomposes the sentence and its annotations (e.g. POS tags, dependency parse) into *substructures*, a word and its associated annotations. Second, it extracts features for each substructure (word), and combines them with the word's embedding to form a *substructure embedding*. Third, we sum over substructure embeddings to form a composed *annotated sentence embedding*, which is used by a final softmax layer to predict the output label (relation). Fourth, it multiplies in the score of each label according to the standard feature-based log-linear model.

The result is a state-of-the-art relation extractor for unseen domains from ACE 2005 (Walker et al., 2006) and the relation classification dataset from SemEval-2010 Task 8 (Hendrickx et al., 2010).

Contributions This chapter makes several contributions, including:

1. We introduce a new hybrid model for relation extraction that combines a log-linear model and the FCM, a compositional embedding model.
2. We obtain the best reported results on ACE-2005 for coarse-grained relation extraction in the cross-domain setting with this model.
3. We obtain results on SemEval-2010 Task 8 competitive with the best reported results.

Note that other work has already been published that builds on the FCM, such as Hashimoto et al. (2015), Nguyen and Grishman (2015), Santos et al. (2015), Yu and Dredze (2015) and Yu et al. (2015). Additionally, the FCM has been extended to incorporate a low-rank embedding of the features (Yu et al., 2015), with a focus on fine-grained relation extraction for ACE and ERE. Here, we obtain better results than the low-rank extension on ACE coarse-grained relation extraction.

4.2 Relation Extraction

In relation extraction we are given a sentence as input with the goal of identifying, for all pairs of entity mentions, what relation exists between them, if any. For each pair of entity mentions in a sentence S, we construct an instance $(y, \mathbf{x})$, where $\mathbf{x} = (M_1, M_2, S, A)$. $S = (w_1, w_2, ..., w_n)$ is a sentence of length n that expresses a relation of type y between two entity mentions M_1 and M_2, where M_1 and M_2 are spans of words in S. A is the associated annotations of sentence S, such as part-of-speech tags, a dependency parse, and named entities. We consider directed relations: for a relation type Rel, $y{=}Rel(M_1, M_2)$ and $y'{=}Rel(M_2, M_1)$ are different relations. Table 4.1 shows ACE 2005 relations, and has a strong label bias towards negative examples. We also consider the task of relation *classification* (SemEval), where the number of negative examples is artificially reduced.

Contrast with Semantic Role Labeling Both relation extraction and relation classification are similar in form to semantic role labeling (SRL), which was the focus of the previous chapter (Chapter 3). As structured prediction tasks, the two tasks look quite similar: in both cases, the goal is to label the relations that hold between a pair of words or a pair of spans. The key difference between them is best explained by an example. Consider the sentence below:

The president was born in Hawaii.

Relation extraction directly identifies the *born-in* relation between "president" and "Hawaii", without explicitly identifying the invoking predicate "born". By contrast, semantic role labeling would explicitly identify the predicate "born" and then label its arguments as the person being born and the location of birth. In this way, SRL identifies the trigger of a relation, whereas relation extraction does not.

Embedding Models Word embeddings and compositional embedding models have been successfully applied to a range of NLP tasks; however, the applications of these embedding models to relation extraction are still limited. Prior work on relation classification (e.g. SemEval 2010 Task 8) has focused on short sentences with at most one relation per sentence (Socher et al., 2012; Zeng et al., 2014). For relation extraction, where negative examples abound, prior work has assumed that only the named entity boundaries and not their types were available (Plank and Moschitti, 2013; Nguyen et al., 2015). Other work has assumed that the order of two entities in a relation is given while the relation type itself is unknown (Nguyen and Grishman, 2014; Nguyen and Grishman, 2015). The standard relation extraction task, as adopted by ACE 2005 (Walker et al., 2006), uses long sentences containing multiple named entities with known types[3] and unknown relation directions. The FCM was the first application of neural language model embeddings to this task.

Motivation and Examples Whether a word is indicative of a relation depends on multiple properties, which may relate to its context within the sentence. For example, whether the word is in-between the entities, on the dependency path between them, or to their left or right may provide additional complementary information. Illustrative examples are given in Table 4.1 and provide the motivation for our model. In the next section, we will show how the FCM develops informative representations capturing both the semantic information in word embeddings and the contextual information expressing a word's role relative to the entity mentions. The FCM was the first model to incorporate all of this information at once. The closest work is that of Nguyen and Grishman (2014), who use a log-linear model for relation extraction with embeddings as features for only the entity heads. Such embedding features are insensitive to the broader contextual information and, as we show, are not sufficient to elicit the word's role in a relation.

[3]Since the focus of this chapter is relation extraction, we adopt the evaluation setting of prior work, which uses gold named entities to better facilitate comparison.

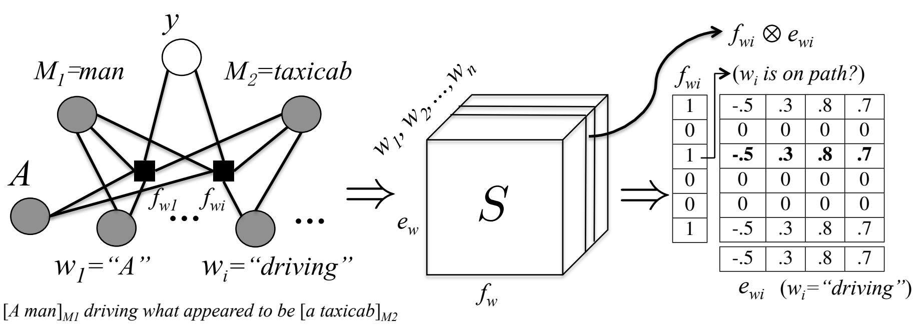


Figure 4.1: Example construction of FCM substructure embeddings. Each substructure is a word w_i in S, augmented by the target entity information and related information from annotation A (e.g. a dependency tree). The diagram shows the factorization of the annotated sentence into substructures (left), the concatenation of the substructure embeddings for the sentence (middle), and a single substructure embedding from that concatenation (right). The annotated sentence embedding (not shown) would be the sum of the substructure embeddings, as opposed to their concatenation.

4.3 Background: Compositional Embedding Model

In this section, we review a general framework to construct an embedding of a sentence with annotations on its component words (Gormley et al., 2015b). While we focus on the relation extraction task, the framework applies to any task that benefits from both embeddings and typical hand-engineered lexical features. We hope to assure the reader of its suitability for the task, just as our results should demonstrate its strong performance in isolation.

4.3.1 Combining Features with Embeddings

We begin by describing a precise method for constructing substructure embeddings and annotated sentence embeddings from existing (usually unlexicalized) features and embeddings. Note that these embeddings can be included directly in a log-linear model as features—doing so results in a special case of the full FCM model presented in the next subsection.

An annotated sentence is first decomposed into substructures. The type of substructures can vary by task; for relation extraction we consider one substructure per word[4]. For each substructure in the sentence we have a handcrafted feature vector f_i and a dense embedding vector e_{w_i}. We represent each substructure as the outer product $\otimes$ between these two vectors to produce a matrix, herein called a **substructure embedding**: $h_i = f_i \otimes e_{w_i}$. The features f_i are based on the local context in S and annotations in A, which can include global information about the annotated sentence. These features allow the model to promote different properties and to distinguish different functions of the words. Feature engineering can be task specific, as relevant annotations can change with regards to each

[4]We use words as substructures for relation extraction, but use the general terminology to maintain model generality.

task. In this work we utilize unlexicalized binary features common in relation extraction. Figure 4.1 depicts the construction of a sentence's substructure embeddings.

We further sum over the substructure embeddings to form an **annotated sentence embedding**:

$$e_x = \sum_{i=1}^{n} f_i \otimes e_{w_i} \tag{4.1}$$

When both the handcrafted features and word embeddings are treated as inputs, as has previously been the case in relation extraction, this annotated sentence embedding can be used directly as features of a log-linear model. In fact, we find that the feature sets used in prior work for many other NLP tasks are special cases of this simple construction (Turian et al., 2010; Nguyen and Grishman, 2014; Hermann et al., 2014; Roth and Woodsend, 2014). This highlights an important connection: when the word embeddings are constant, the constructions of substructure and annotated sentence embeddings are just specific forms of polynomial (specifically quadratic) feature combination—hence their commonality in the literature. The K dimensions of e_w specify the strengths of K non-binary features of word w. The experimental results suggest that such a construction is more powerful than directly including embeddings into the model (i.e. without the outer-product with the features).

4.3.2 The Log-Bilinear Model

The full log-bilinear model first forms the substructure and annotated sentence embeddings from the previous subsection. The model uses its parameters to score the annotated sentence embedding and uses a softmax to produce an output label. We call the entire model the **Feature-rich Compositional Embedding Model (FCM)**.

Our task is to determine the label y (relation) given the instance $\mathbf{x} = (M_1, M_2, S, A)$. We formulate this as a probability.

$$p_{\text{FCM}}(y|\mathbf{x}; T, \mathbf{e}) = \frac{\exp\left(\sum_{i=1}^{n} \langle T_y, f_{w_i} \otimes e_{w_i} \rangle_F\right)}{Z(\mathbf{x})} \tag{4.2}$$

where $\langle \cdot, \cdot \rangle_F$ is the 'matrix dot product' or Frobenius inner product of the two matrices. The normalizing constant which sums over all possible output labels $y' \in L$ is given by $Z(\mathbf{x}) = \sum_{y' \in L} \exp\left(\sum_{i=1}^{n} \langle T_{y'}, f_{w_i} \otimes e_{w_i} \rangle_F\right)$. The parameters of the model are the word embeddings $\mathbf{e}$ for each word type and a list of weight matrices $T = [T_y]_{y \in L}$ which is used to score each label y. The model is log-bilinear (i.e. log-quadratic) since we recover a log-*linear* model by fixing either $\mathbf{e}$ or T. This chapter studies both the full log-bilinear and the log-linear model obtained by treating the word embeddings as constant rather than as parameters of the model.

Other popular log-bilinear models are the log-bilinear language models (Mnih and Hinton, 2007; Mikolov et al., 2013). The Log-Bilinear Language Model (LBLM) of Mnih and Hinton (2007) provides a model $p(w_n|w_1, w_2, \ldots, w_{n-1})$ where $w_1, w_2, \ldots, w_{n-1}$ are the context words and w_n is the predicted word. In the LBLM, the bilinear interaction is between the word embeddings of the context words and the embedding vector of the predicted word. By contrast, the bilinear interaction in our FCM model is between the word embeddings e_{w_i} and the parameter tensor T_y as described above.

4.3.3 Discussion of the Compositional Model

Substructure Embeddings Similar words (i.e. those with similar embeddings) with similar functions in the sentence (i.e. those with similar features) will have similar matrix representations. To understand the selection of the outer product, consider the example in Fig. 4.1. The word "driving" can indicate the *ART* relation if it appears on the dependency path between M_1 and M_2. Suppose the third feature in f_{w_i} indicates this `on-path` feature. The FCM can now learn parameters that give the third row a high weight for the *ART* label. Other words with embeddings similar to "driving" that appear on the dependency path between the mentions will similarly receive high weight for the *ART* label. On the other hand, if the embedding is similar but is not on the dependency path, it will have 0 weight. Thus, the model generalizes its model parameters across words with similar embeddings only when they share similar functions in the sentence.

Smoothed Lexical Features Another intuition about the selection of outer product is that it is actually a smoothed version of traditional lexical features used in classical NLP systems. Consider a lexical feature $f = u \wedge w$, which is a conjunction (logic-and) between non-lexical property u and lexical part (word) w. If we represent w as a one-hot vector, then the outer product exactly recovers the original feature f. Then if we replace the one-hot representation with its word embedding, we get the current form of the FCM. Therefore, the model can be viewed as a smoothed version of lexical features, which keeps the expressive strength, and uses embeddings to generalize to low frequency features.

Time Complexity Inference in FCM is much faster than both CNNs (Collobert et al., 2011b) and RNNs (Socher et al., 2013b; Bordes et al., 2012). FCM requires $O(nds)$ products on average, where n is the length of the sentence, d is the dimension of word embedding, and s is the average number of per-word non-zero feature values. That is, $s = \frac{1}{n}\sum_{i=1}^{n}\sum_{j} f_{i,j}$ where f_i in our models is a sparse binary feature vector. In contrast, CNNs and RNNs usually have complexity $O(C \cdot nd^2)$, where C is a model dependent constant.

4.4 A Log-linear Model

Our log-linear model uses a rich binary feature set from Sun et al. (2011) (Baseline)—this consists of all the baseline features of Zhou et al. (2005) plus several additional carefully-chosen features that have been highly tuned for ACE-style relation extraction over years of research. We exclude the Country gazetteer and WordNet features from Zhou et al. (2005). For a detailed description of the features, we direct the reader to Zhou et al. (2005) and Sun et al. (2011). Here, we provide a summary of the types of context considered by them:

- The words of each mention, their head words, and combinations of these
- Words in between the mentions plus indicators of the number of words intervening
- Words appearing immediately before or after the mentions

- Entity types and phrase types of the mentions
- Counts of the number of intervening mentions
- Indicators for whether the mentions overlap and their direction
- Features based on the heads of the chunks intervening between the mentions, before or after the mentions
- Combinations of labels of the chunks between the mentions
- Features combining information from a dependency tree (e.g. head of mention, dependent of mention) with entity type information
- Features combining information from a constituency tree (e.g. is head contained within an NP) with entity type information
- Labels along shortest path through constituency tree
- Bigrams of the words in between the mentions
- The full sequence of words in between the mentions
- Labels of a high cut through the constituency tree
- Parts-of-speech, words, or labels of the shortest dependency tree path between the mentions

The features incorporate information from entity types, mention types, parts-of-speech, a dependency tree, constituency tree, and chunking of the sentence. These features differ from those used for SRL in Section 3.2.5. The reason for the difference is that we selected the features from the prior state-of-the-art methods for each task; as discussed in Section 4.2, the two tasks have qualitative differences and the literature has typically treated the two tasks as distinct.

The log-linear model has the usual form:

$$p_{\text{loglin}}(y|\boldsymbol{x}) \propto \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y)) \tag{4.3}$$

where $\boldsymbol{\theta}$ are the model parameters and $\boldsymbol{f}(\boldsymbol{x}, y)$ is a vector of features.

4.5 Hybrid Model

We present a product-of-experts model (Hinton, 2002), which combines the FCM (Section 4.3.2) with an existing log-linear model (Section 4.4). We do so by defining a new model:

$$p_{\text{FCM+loglin}}(y|\boldsymbol{x}; T, \mathrm{e}, \boldsymbol{\theta}) = \frac{1}{Z} p_{\text{FCM}}(y|\boldsymbol{x}; T, \mathrm{e})\, p_{\text{loglin}}(y|\boldsymbol{x}; \boldsymbol{\theta}) \tag{4.4}$$

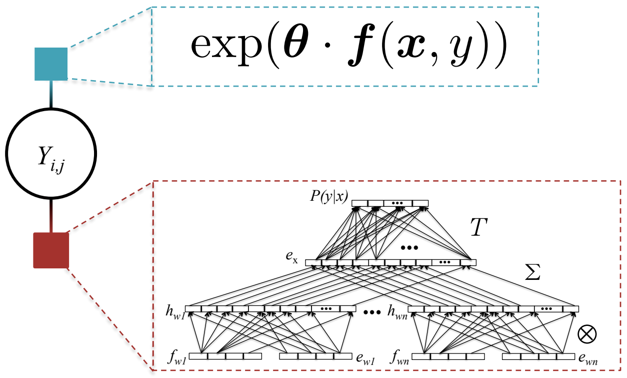


Figure 4.2: Factor graph of the hybrid model. The variable $Y_{i,j}$ ranges over possible relations for the ith and jth entity mentions in the sentence. The top factor (blue) multiplies in the score according to a log-linear model. The bottom factor (red) multiplies in the score of the FCM, a compositional embedding model—depicted here as a neural network.

The integration treats each model as providing a score which we multiply together. The constant Z ensures a normalized distribution. In this hybrid model, we assume that the features for p_{FCM} and p_{loglin} are different, with the latter generally using a much richer feature set. If the two submodels use the same features, then the hybrid model reduces to an FCM with e_w having an additional dimension that always takes value 1. We can view this as a very simple factor graph (Section 2.3.1) consisting of just one variable and two factors—corresponding to the two submodels. This representation of our hybrid model is shown in Figure 4.2. In this chapter, we do not make use of this factor graph representation, except as a pedagogical tool for understanding the model. However, in a subsequent chapter, (Chapter 6), we will consider more complicated hybrid models for which approximate inference will be run over the factor graph.

To train we optimize a conditional log-likelihood objective (Section 2.3.4.1):

$$\ell(D; T, \mathbf{e}, \boldsymbol{\theta}) = \sum_{(\mathbf{x},y)\in D} \log p(y|\mathbf{x}; T, \mathbf{e}, \boldsymbol{\theta})$$

where D is the set of all training data, $\mathbf{e}$ is the set of word embeddings, T is the FCM tensor parameters, and $\boldsymbol{\theta}$ are the parameters of the log-linear model. To optimize the objective, for each instance $(y, \mathbf{x})$ we perform stochastic training on the loss function $\ell = \ell(y, \mathbf{x}; T, \mathbf{e}, \boldsymbol{\theta}) = \log p(y|\mathbf{x}; T, \mathbf{e}, \boldsymbol{\theta})$.

The gradients of the model parameters are obtained by backpropagation (Section 2.2.2) (i.e. repeated application of the chain rule). For the hybrid model, this is easily computed since each sub-model has separate parameters. When we treat the word embeddings as parameters (i.e. the log-bilinear FCM), we also fine-tune the word embeddings with the FCM model. As is common in deep learning, we initialize these embeddings from a neural language model and then *fine-tune* them for our supervised task.

Set	**Template**
`HeadEmb`	$\{I[i = h_1], I[i = h_2]\}$ (w_i is head of M_1/M_2) $\times\{\phi, t_{h_1}, t_{h_2}, t_{h_1} \oplus t_{h_2}\}$
`Context`	$I[i = h_1 \pm 1]$ (left/right token of w_{h_1}) $I[i = h_2 \pm 1]$ (left/right token of w_{h_2})
`In-between`	$I[i > h_1]\&I[i < h_2]$ (in between) $\times\{\phi, t_{h_1}, t_{h_2}, t_{h_1} \oplus t_{h_2}\}$
`On-path`	$I[w_i \in P]$ (on path) $\times\{\phi, t_{h_1}, t_{h_2}, t_{h_1} \oplus t_{h_2}\}$

Table 4.2: Feature sets used in FCM.

4.6 Main Experiments

Our primary experiments consider two settings: relation extraction on ACE 2005 and relation classification on SemEval-2010 Task 8.

4.6.1 Experimental Settings

Features Our FCM features (Table 4.2) use a feature vector f_{w_i} over the word w_i, the two target entities M_1, M_2, and their dependency path. Here h_1, h_2 are the indices of the two head words of M_1, M_2, $\times$ refers to the Cartesian product between two sets, t_{h_1} and t_{h_2} are entity types (named entity tags for ACE 2005 or WordNet supertags for SemEval 2010—see Table 4.3 for example tags) of the head words of two entities, and ϕ stands for the empty feature. $\oplus$ refers to the conjunction of two elements. The `In-between` features indicate whether a word w_i is in between two target entities, and the `On-path` features indicate whether the word is on the dependency path, on which there is a set of words P, between the two entities.

We also use the target entity type as a feature. Combining this with the basic features results in more powerful compound features, which can help us better distinguish the functions of word embeddings for predicting certain relations. For example, if we have a person and a vehicle, we know it will be more likely that they have an *ART* relation. For the *ART* relation, we introduce a corresponding weight vector, which is closer to lexical embeddings similar to the embedding of "drive".

All linguistic annotations needed for features (POS, chunks[5], parses) are from Stanford CoreNLP (Manning et al., 2014). Since SemEval does not have gold entity types we obtained WordNet and named entity tags using Ciaramita and Altun (2006). For all experiments we use 200-d word embeddings trained on the NYT portion of the Gigaword 5.0 corpus (Parker et al., 2011). The embeddings are trained with word2vec[6] using the CBOW model with negative sampling (15 negative words) (Mikolov et al., 2013). We set a window size c=5, and remove types occurring less than 5 times.

[5]Obtained from the constituency parse using the CONLL 2000 chunking converter (Perl script).
[6]`https://code.google.com/p/word2vec/`

Models We consider several methods. The first three methods (A, B, C) are examples of either the FCM alone or a standard log-linear model. The two remaining methods (D, E) are hybrid models that integrate FCM as a submodel within the hybrid model (Section 4.5)—we consider two such combinations.

(A) FCM (Section 4.3) in isolation without fine-tuning. We use the complete feature set given in Table 4.2.

(B) FCM (Section 4.3.2) in isolation with fine-tuning (i.e. trained as a log-bilinear model), using the same feature set as in (A).

(C) The log-linear model alone (Section 4.4).

(D) A hybrid model combines a restricted form of the FCM with the log-linear model from (C) above. For the FCM, we restrict to the feature set of Nguyen and Grishman (2014) obtained by using the embeddings of heads of two entity mentions (+HeadOnly).

(E) Our full hybrid model which combines the FCM from (A) with the log-linear model from (C) above.

All models use L2 regularization tuned on dev data.

Datasets and Evaluation

ACE 2005 We evaluate our relation extraction system on the English portion of the ACE 2005 corpus (Walker et al., 2006).[7] There are 6 domains: Newswire (`nw`), Broadcast Conversation (`bc`), Broadcast News (`bn`), Telephone Speech (`cts`), Usenet Newsgroups (`un`), and Weblogs (`wl`). Following prior work we focus on the domain adaptation setting, where we train on one set (the union of the news domains (`bn+nw`), tune hyperparameters[8] on a dev domain (half of `bc`) and evaluate on the remainder (`cts`, `wl`, and the remainder of `bc`) (Plank and Moschitti, 2013; Nguyen and Grishman, 2014). The LDC release of the ACE data contains four distinct annotations: fp1, fp2, adj, timex2norm. Following Plank and Moschitti (2013), we use the adjudicated fileset (adj) – these are files which were annotated twice and for which discrepancies were resolved.

We assume that gold entity spans *and* types are available for train and test. We use all pairs of entity mentions to yield 43,497 total relations in the training set, of which 3,658 are non-nil. One curious aspect of the ACE data is that some relations are self-referential. That is, the pair of entities is some entity and itself. We also included these self-referential

[7]Many relation extraction systems evaluate on the ACE 2004 corpus (Mitchell et al., 2005). Unfortunately, the most common convention is to use 5-fold cross validation, treating the entirety of the dataset as both train *and* evaluation data. Rather than continuing to overfit this data by perpetuating the cross-validation convention, we instead focus on ACE 2005.

[8]For each ACE 2005 model, we performed a grid-search over hyperparameters and selected the model which obtained the highest F1 on the development set. There were four hyperparameters tuned by the grid search: (1) the variance of the L2 regularizer $\sigma^2 \in \{40000, 400000\}$, (2) a constant $\gamma \in \{0.1, 1, 10\}$ used to scale the initial embeddings after they were renormalized to sum-to-one, (3) the AdaGrad learning rate $\eta \in \{0.01, 0.1\}$, and (4) AdaGrad's initial value for the sum of the squares $\delta \in \{0.1, 1\}$.

relations so that the number of non-nil relations would be identical to that reported in the original ACE dataset. We did not include any negative examples of self-referential relations. We followed Table 6 of the ACE 2005 annotation guidelines to determine which of the relations should be treated as symmetric (METONYMY, PER-SOC, and PHYS) and asymmetric (ART, GEN-AFF, ORG-AFF, and PART-WHOLE). The nil relation is treated as symmetric. Thus, the total output space for our models would be 12 labels, but the METONYMY relation never appears in any explicit relation *mentions* in the ACE dataset. So the total number of observed labels in the training data is only 11. We report precision, recall, and micro F1 for relation extraction. While it is not our focus, for completeness we include results with unknown entity types following Plank and Moschitti (2013) (Section 4.7).

SemEval 2010 Task 8 We evaluate on the SemEval 2010 Task 8 dataset[9] (Hendrickx et al., 2010) to compare with other compositional models and highlight the advantages of our models. This task is to determine the relation type (or no relation) between two entities in a sentence. We adopt the setting of Socher et al. (2012). We use 10-fold cross validation on the training data to select hyperparameters and do regularization by early stopping. The learning rates for FCM with/without fine-tuning are 5e-3 and 5e-2 respectively. We report macro-F1 and compare to previously published results.

As noted earlier, we distinguish between two tasks: ACE 2005 relation *extraction* and SemEval 2010 Task 8 relation *classification*. The key distinction between them is the proportion of entity pairs that are labeled as having no relation. In the ACE 2005 training set, only 10.1% of training instances are non-nil relations, the rest are nil. In the SemEval data, 82.6% of the instances are labeled with one of the 9 standard relations and 17.4% relations are labeled as Other (a category which could include nil relations).

4.6.2 Results

ACE 2005 Despite FCM's (A) simple feature set (see Table 4.2), it is competitive with the log-linear baseline (C) on out-of-domain test sets (Table 4.4). In the typical gold entity spans and types setting, both Plank and Moschitti (2013) and Nguyen and Grishman (2014) found that they were unable to obtain improvements by adding embeddings to baseline feature sets. By contrast, we find that on all domains the combination baseline + FCM (E) obtains the highest F1 and significantly outperforms the other baselines, *yielding the best reported results for this task.* We found that fine-tuning of embeddings (B) did not yield improvements on our out-of-domain development set, in contrast to our results below for SemEval. We suspect this is because fine-tuning allows the model to overfit the training domain, which then hurts performance on the unseen ACE test domains. Accordingly, Table 4.4 shows only the log-linear model.

Finally, we highlight an important contrast between FCM (A) and the log-linear model (C): the latter uses over 50 feature templates based on a POS tagger, dependency parser, chunker, and constituency parser. The resulting model has about 6.5 million parameters.

[9] http://docs.google.com/View?docid=dfvxd49s_36c28v9pmw

	Tag Type	# Tags	Example Tags
ACE 2005	Types	7	FAC, GPE, LOC, ORG, PER, VEH, WEA
	Subtypes	43	FAC:Airport, FAC:Building-Grounds, FAC:Path, FAC:Plant, FAC:Subarea-Facility, GPE:Continent, GPE:County-or-District, GPE:GPE-Cluster, GPE:Nation, GPE:Population-Center, . . .
SemEval 2010 Task 8	NER	50	E_ANIMAL, E_DISEASE, E_EVENT_OTHER, E_FAC_DESC_AIRPORT, E_FAC_DESC_BRIDGE, E_FAC_DESC_BUILDING, E_FAC_DESC_HIGHWAY_STREET, E_FAC_DESC_OTHER, E_GAME, E_GPE_CITY, . . .
	WordNet	46	adj.all, adj.pert, adj.ppl, adv.all, noun.Tops, noun.act, noun.animal, noun.artifact, noun.attribute, noun.body, . . .

Table 4.3: Named entity tags for ACE 2005 and SemEval 2010 Task 8. The ACE 2005 subtypes are only shown for reference: all of our features referred to the types only.

FCM uses only a dependency parse and about 0.5 million parameters (not including the embeddings, which are held constant) but still obtains better results (Avg. F1).

SemEval 2010 Task 8 Table 4.5 compares our models to the best reported results from the SemEval-2010 Task 8 shared task and several other compositional models.

For the FCM we considered two feature sets. We found that using NE tags instead of WordNet tags helps with fine-tuning but hurts without. This may be because the set of WordNet tags is larger making the model more expressive, but also introduces more parameters. When the embeddings are fixed, they can help to better distinguish different functions of embeddings. But when fine-tuning, it becomes easier to over-fit. Alleviating over-fitting is a subject for future work (Section 4.9).

With either WordNet or NER features, FCM achieves better performance than the RNN and MVRNN. With NER features and fine-tuning, it outperforms a CNN (Zeng et al., 2014) and also the combination of an embedding model and a traditional log-linear model (RNN/MVRNN + linear) (Socher et al., 2012). As with ACE, FCM uses less linguistic resources than many close competitors (Rink and Harabagiu, 2010).

We also compared to concurrent work on enhancing the compositional models with task-specific information for relation classification, including Hashimoto et al. (2015) (RelEmb), which trained task-specific word embeddings, and Santos et al. (2015) (CR-CNN), which proposed a task-specific ranking-based loss function. Our Hybrid methods (FCM + linear) get comparable results to theirs. Note that their base compositional model results without any task-specific enhancements, i.e. RelEmb with word2vec embeddings and CR-CNN

	Model	bc			cts			wl			Avg.
		P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
(A)	FCM-only (ST)	66.56	**57.86**	61.90	65.62	44.35	52.93	57.80	44.62	50.36	55.06
(C)	Baseline (ST)	**74.89**	48.54	58.90	74.32	40.26	52.23	63.41	43.20	51.39	54.17
(D)	+ HeadOnly (ST)	70.87	50.76	59.16	71.16	43.21	53.77	57.71	42.92	49.23	54.05
(E)	+ FCM (ST)	74.39	55.35	**63.48**	**74.53**	**45.01**	**56.12**	**65.63**	**47.59**	**55.17**	**58.26**

Table 4.4: Comparison of models on ACE 2005 out-of-domain test sets. Baseline + HeadOnly is our reimplementation of the features of Nguyen and Grishman (2014).

Classifier	**Features**	**F1**
SVM (Rink and Harabagiu, 2010) (Best in SemEval2010)	POS, prefixes, morphological, WordNet, dependency parse, Levin classed, ProBank, FrameNet, NomLex-Plus, Google n-gram, paraphrases, TextRunner	82.2
RNN	word embedding, syntactic parse	74.8
RNN + linear	word embedding, syntactic parse, POS, NER, WordNet	77.6
MVRNN	word embedding, syntactic parse	79.1
MVRNN + linear	word embedding, syntactic parse, POS, NER, WordNet	82.4
CNN (Zeng et al., 2014)	word embedding, WordNet	82.7
CR-CNN (log-loss)	word embedding	82.7
CR-CNN (ranking-loss)	word embedding	**84.1**
RelEmb (word2vec embedding)	word embedding	81.8
RelEmb (task-spec embedding)	word embedding	82.8
RelEmb (task-spec embedding) + linear	word embedding, dependency paths, WordNet, NE	83.5
DepNN	word embedding, dependency paths	82.8
DepNN + linear	word embedding, dependency paths, WordNet, NER	83.6
(A) FCM (log-linear)	word embedding, dependency parse, WordNet	82.0
	word embedding, dependency parse, NER	81.4
(B) FCM (log-bilinear)	word embedding, dependency parse, WordNet	82.5
	word embedding, dependency parse, NER	83.0
(E) FCM (log-linear) + linear (Hybrid)	word embedding, dependency parse, WordNet	83.1
	word embedding, dependency parse, NER	83.4

Table 4.5: Comparison of our models with previously published results for SemEval 2010 Task 8.

with log-loss, are still lower than the best FCM result. Our main finding is that the hybrid model again performs better than either of its submodels alone.

Finally, a concurrent work (Liu et al., 2015) proposes DepNN, which builds representations for the dependency path (and its attached subtrees) between two entities by applying recursive and convolutional neural networks successively. Compared to their model, the FCM achieves comparable results. Of note, the FCM and the RelEmb are also the most efficient models among all above compositional models since they have linear time complexity with respect to the dimension of embeddings.

4.7 Additional ACE 2005 Experiments

Next we present results in a distinct setting for ACE 2005 in which the gold entity types are not available. This allows for additional comparison with prior work (Plank and Moschitti, 2013).

4.7.1 Experimental Settings

Data For comparison with Plank and Moschitti (2013), we (1) generate relation instances from all pairs of entities within each sentence with three or fewer intervening entity mentions—labeling those pairs with no relation as negative instances, (2) use gold entity spans (but not types) at train and test time, and (3) evaluate on the 7 coarse relation types, ignoring the subtypes. In the training set, 34,669 total relations are annotated of which only 3,658 are non-nil relations. We did not match the number of tokens they reported in the `cts` and `wl` domains. Therefore, in this section we only report the results on the test set of `bc` domain.

Models and Features We run the same models as in Section 4.6.2 on this task. Here the FCM does not use entity type features. Plank and Moschitti (2013) also use Brown clusters and word vectors learned by latent-semantic analysis (LSA). In order to make a fair comparison with their method, we also report the FCM result using Brown clusters (prefixes of length 5) of entity heads as entity types. Furthermore, we report non-comparable settings using WordNet super-sense tags of entity heads as types. The WordNet features were also used in their paper but not as substitution of entity types. We use the same toolkit to get the WordNet tags as in Section 4.6.1. The Brown clusters are from (Koo et al., 2008)[10].

4.7.2 Results

Table 4.6 shows the results under the low-resource setting. When no entity types are available, the performance of the FCM-only model greatly decreases to 48.15%, which is consistent with our observation in the ablation tests. The baseline model also relies heavily on the entity types: without entity type information the performance of our baseline model drop to 40.62%, even lower than the reduced FCM only model.

[10] `http://people.csail.mit.edu/maestro/papers/bllip-clusters.gz`

	bc		
Model	**P**	**R**	**F1**
PM'13 (Brown)	54.4	43.4	48.3
PM'13 (LSA)	53.9	45.2	49.2
PM'13 (Combination)	55.3	43.1	48.5
(A) FCM only	53.7	43.7	48.2
(C) Baseline	59.4	30.9	40.6
(E) + HeadOnly	64.9	41.3	50.5
(E) + FCM	**65.5**	41.5	50.8
(A) FCM only w/ Brown	64.6	40.2	49.6
(A) FCM only w/WordNet	64.0	43.2	51.6
Linear+Emb	46.5	**49.3**	47.8
Tree-kernel+Emb (Single)	57.6	46.6	51.5
Tree-kernel+Emb (Combination)	58.5	47.3	**52.3**

Table 4.6: Comparison of models on ACE 2005 out-of-domain test sets for the low-resource setting, where the gold entity spans are known but entity types are unknown. PM'13 is the results reported in Plank and Moschitti (2013). "Linear+Emb" is the implementation of our method (4) in (Nguyen et al., 2015). The "Tree-kernel+Emb" methods are the enrichments of tree-kernels with embeddings proposed by Nguyen et al. (2015).

The combination of baseline model and head embeddings (Baseline + HeadOnly) greatly improve the results. This is consistent with the observation in Nguyen and Grishman (2014) that when the gold entity types are unknown, information of the entity heads provided by their embeddings will play a more important role. Combination of the baseline and FCM (Baseline + FCM) also achieves improvement but not significantly better than Baseline + HeadOnly. A possible explanation is that FCM becomes less efficient on using context word embeddings when the entity type information is unavailable. In this situation the head embeddings provided by FCM become the dominating contribution to the baseline model, making the model have similar behavior as the Baseline + HeadOnly method.

Finally, we find Brown clusters can help FCM when entity types are unknown. Although the performance is still not significantly better than Baseline + HeadOnly, it outperforms all the results in Plank and Moschitti (2013) as a *single model*, and with the *same source of features*. WordNet super-sense tags further improve FCM, and achieves the best reported results on this low-resource setting. These results are encouraging since it shows FCM may be more useful under the end-to-end setting where predictions of both entity mentions and relation mentions are required in place of predicting relation based on gold tags (Li and Ji, 2014).

Recently Nguyen et al. (2015) proposed a novel way of applying embeddings to tree-kernels. From the results, our best single model achieves comparable result with their best single system, while their combination method is slightly better than ours. This suggests that we may benefit more from combining the usages of multiple word representations; and we will investigate it in future work.

4.8 Related Work

Compositional Models for Sentences In order to build a representation (embedding) for a sentence based on its component word embeddings and structural information, recent work on compositional models (stemming from the deep learning community) has designed model structures that mimic the structure of the input. For example, these models could take into account the order of the words (as in Convolutional Neural Networks (CNNs)) (Collobert et al., 2011b) or build off of an input tree (as in Recursive Neural Networks (RNNs) or the Semantic Matching Energy Function) (Socher et al., 2013b; Bordes et al., 2012).

While these models work well on sentence-level representations, the nature of their designs also limits them to fixed types of substructures from the annotated sentence, such as chains for CNNs and trees for RNNs. Such models cannot capture arbitrary combinations of linguistic annotations available for a given task, such as word order, dependency tree, and named entities used for relation extraction. Moreover, these approaches ignore the differences in functions between words appearing in different roles. This does not suit more general substructure labeling tasks in NLP, e.g. these models cannot be directly applied to relation extraction since they will output the same result for any pair of entities in a same sentence. By contrast, the FCM annotates the sentence with the entity pair before embedding it, so that the embedding is of the entire annotated sentence.

Compositional Models with Annotation Features To tackle the problem of traditional compositional models, Socher et al. (2012) made the RNN model specific to relation extraction tasks by working on the minimal subtree that spans the two target entities. However, these specializations to relation extraction do not generalize easily to other tasks in NLP. There are two ways to achieve such specialization in a more general fashion:

1. Enhancing Compositional Models with Features. A recent trend enhances compositional models with annotation features. Such an approach has been shown to significantly improve over pure compositional models. For example, Hermann et al. (2014) and Nguyen and Grishman (2014) gave different weights to words with different syntactic context types or to entity head words with different argument IDs. Zeng et al. (2014) use concatenations of embeddings as features in a CNN model, according to their positions relative to the target entity mentions. Belinkov et al. (2014) enrich embeddings with linguistic features before feeding them forward to a RNN model. Socher et al. (2013a) and Hermann and Blunsom (2013) enhanced RNN models by refining the transformation matrices with phrase types and CCG super tags.

2. Engineering of Embedding Features. A different approach to combining traditional linguistic features and embeddings is hand-engineering features with word embeddings and adding them to log-linear models. Such approaches have achieved state-of-the-art results in many tasks including NER, chunking, dependency parsing, semantic role labeling, and relation extraction (Miller et al., 2004; Turian et al., 2010; Koo et al., 2008; Roth and Woodsend, 2014; Sun et al., 2011; Plank and Moschitti, 2013). Roth and Woodsend (2014) considered features similar to ours for semantic role labeling.

However, in prior work both of above approaches are only able to utilize limited information, usually one property for each word. Yet there may be different useful properties of a word that can contribute to the performances of the task. By contrast, our model can easily utilize these features without changing the model structures.

Task-Specific Enhancements for Relation Classification An orthogonal direction of improving compositional models for relation classification is to enhance the models with task-specific information. For example, Hashimoto et al. (2015) trained task-specific word embeddings, and Santos et al. (2015) proposed a ranking-based loss function for relation classification.

4.9 Summary

We have presented a new hybrid model for combining a log-linear model with the FCM, a compositional model for deriving sentence-level and substructure embeddings from word embeddings. Compared to existing compositional models, our hybrid model can easily handle arbitrary types of input and handle global information for composition, while remaining easy to implement. We have demonstrated that the compositional model FCM alone attains near state-of-the-art performances on several relation extraction tasks, and in combination with traditional feature based log-linear models it obtains state-of-the-art results. On an cross-domain setting for ACE 2005, the FCM obtains 55.06 Avg. F1 and the hybrid gets 58.26 Avg. F1. On SemEval 2010 Task 8, the FCM gets 83.0 F1 and the hybrid model obtains 83.4 F1.

Chapter 5

Approximation-aware Learning for Structured Belief Propagation

Having motivated the use of latent variables, structured factors, and neural factors, we turn to the remaining problem: learning with inexact inference. Of course, it is possible to build effective models without resorting to inexact inference—the previous two chapters exemplified this fact. However, joint modeling is fundamentally about enabling factors that express opinions about wider contexts of variables. Doing so is what leads to the sort of high treewidth models that require approximate inference.

This chapter[1] develops a learning framework that will cope with inexact *marginal* inference in the types of structured models that we care about. Viewed under a different lens, this chapter is about defining new *models* that resemble neural networks whose topology is inspired by structured belief propagation run on a graphical model. Though joint modeling is our end goal, we currently consider a simpler class of models for which approximate inference is fast, but for which we also have efficient exact inference algorithms. This allows us to better study the behavior of our new learning algorithm for structured belief propagation.

We show how to train the fast dependency parser of Smith and Eisner (2008) for improved accuracy. This parser can consider higher-order interactions among edges while retaining $O(n^3)$ runtime. It outputs the parse with maximum expected recall—but for speed, this expectation is taken under a posterior distribution that is constructed only approximately, using loopy belief propagation through structured factors. We show how to adjust the model parameters to compensate for the errors introduced by this approximation, by following the gradient of the actual loss on training data. We find this gradient by back-propagation. That is, we treat the entire parser (approximations and all) as a differentiable circuit, as others have done for loopy CRFs (Domke, 2010; Stoyanov et al., 2011; Domke, 2011; Stoyanov and Eisner, 2012). The resulting parser obtains higher accuracy with fewer iterations of belief propagation than one trained by conditional log-likelihood.

[1]A previous version of this work was presented in Gormley et al. (2015a).

5.1 Introduction

Recent improvements to dependency parsing accuracy have been driven by higher-order features. Such a feature can look beyond just the parent and child words connected by a single edge to also consider siblings, grandparents, etc. By including increasingly global information, these features provide more information for the parser—but they also complicate inference. The resulting higher-order parsers depend on *approximate* inference and decoding procedures, which may prevent them from predicting the best parse.

For example, consider the dependency parser we will train in this chapter, which is based on the work of Smith and Eisner (2008). Ostensibly, this parser finds the minimum Bayes risk (MBR) parse under a probability distribution defined by a higher-order dependency parsing model. In reality, it achieves $O(n^3 t_{\max})$ runtime by relying on three *approximations during inference*: (1) variational inference by loopy belief propagation (BP) on a factor graph, (2) truncating inference after $t_{\max}$ iterations prior to convergence, and (3) a first-order pruning model to limit the number of edges considered in the higher-order model. Such parsers are traditionally trained *as if the inference had been exact*.[2]

In contrast, we train the parser such that the *approximate* system performs well on the final evaluation function. We treat the entire parsing computation as a differentiable circuit, and backpropagate the evaluation function through our approximate inference and decoding methods to improve its parameters by gradient descent. The system also learns to cope with model misspecification, where the model couldn't perfectly fit the distribution even absent the approximations. For standard graphical models, Stoyanov and Eisner (2012) call this approach ERMA, for "empirical risk minimization under approximations." For objectives besides empirical risk, Domke (2011) refers to it as "learning with truncated message passing."[3]

Our primary contribution is the application of this approximation-aware learning method in the parsing setting, for which the graphical model involves a *global constraint*. Smith and Eisner (2008) previously showed how to run BP in this setting (by calling the inside-outside algorithm as a subroutine). We must backpropagate the downstream objective function through their algorithm so that we can follow its gradient. We carefully define an empirical risk objective function (à la ERMA) to be smooth and differentiable, yet equivalent to accuracy of the minimum Bayes risk (MBR) parse in the limit. Finding this difficult to optimize, we introduce a new simpler objective function based on the L_2 distance between the approximate marginals and the "true" marginals from the gold data.

The goal of this work is to account for the approximations made by a system rooted in structured belief propagation. Taking such approximations into account during training enables us to improve the speed and accuracy of inference at test time. We compare our training method with the standard approach of conditional log-likelihood (CLL) training.

[2]For perceptron training, utilizing inexact inference as a drop-in replacement for exact inference can badly mislead the learner (Kulesza and Pereira, 2008; Huang et al., 2012).

[3]A related approach that we do not explore here are the BP Inference Machines (BPIM) of Ross et al. (2011). Unlike ERMA (Stoyanov et al., 2011) and truncated message passing (Domke, 2011), BPIMs dispense with the standard message passing equations and instead train a sequence of logistic regressors to output accurate messages directly. Like our work, BPIMs define a differentiable computation and the approximate system can be trained end-to-end with backpropagation.

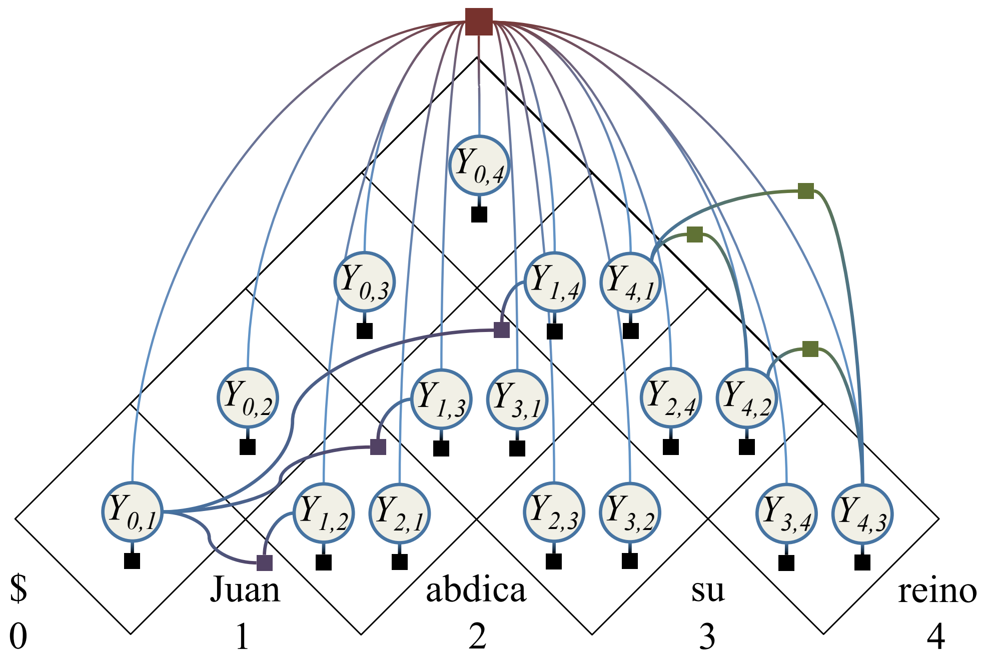


Figure 5.1: Factor graph for dependency parsing of a 4-word sentence; $ is the root of the dependency graph. The boolean variable $Y_{h,m}$ encodes whether the edge from parent h to child m is present. The unary factor (black) connected to this variable scores the edge in isolation (given the sentence). The PTREE factor (red) coordinates all variables to ensure that the edges form a tree. The drawing shows a few higher-order factors (purple for grandparents, green for arbitrary siblings); these are responsible for the graph being cyclic ("loopy").

We evaluate our parser on 19 languages from the CoNLL-2006 (Buchholz and Marsi, 2006) and CoNLL-2007 (Nivre et al., 2007) Shared Tasks as well as the English Penn Treebank (Marcus et al., 1993). On English, the resulting parser obtains higher accuracy with fewer iterations of BP than CLL. On the CoNLL languages, we find that on average it yields higher accuracy parsers than CLL, particularly when limited to few BP iterations.

5.2 Dependency Parsing by Belief Propagation

This section describes the parser that we will train.

Model A factor graph (Frey et al., 1997; Kschischang et al., 2001) (as described in Section 2.3.1) defines the factorization of a probability distribution over a set of variables $\{Y_1, Y_2, \ldots\}$. It is a bipartite graph between variables Y_i and factors α. Edges connect each factor α to a subset of the variables $\{Y_{\alpha_1}, Y_{\alpha_2}, \ldots\}$, called its neighbors. Each factor defines a potential function ψ_α, which assigns a nonnegative score to each configuration of its neighbors $\boldsymbol{y}_\alpha = \{y_{\alpha_1}, y_{\alpha_2}, \ldots\}$. We define the probability of a given assignment $\boldsymbol{y} = \{y_1, y_2, \ldots\}$ to be proportional to the product of all factors' potential functions: $p(\boldsymbol{y}) = \frac{1}{Z} \prod_\alpha \psi_\alpha(\boldsymbol{y}_\alpha)$.

Smith and Eisner (2008) define a factor graph for dependency parsing of a given n-word sentence: n^2 binary variables indicate which of the directed arcs are included ($y_i = \text{ON}$) or excluded ($y_i = \text{OFF}$) in the dependency parse. One of the factors plays the role of a hard global constraint: $\psi_{\text{PTREE}}(\boldsymbol{y})$ is 1 or 0 according to whether the assignment encodes a projective dependency tree. Another n^2 factors (one per variable) evaluate the individual arcs given the sentence, so that $p(\boldsymbol{y})$ describes a first-order dependency parser. A higher-order parsing model is achieved by also including higher-order factors, each scoring configurations of two or more arcs, such as grandparent and sibling configurations. Higher-order factors tend to create cycles in the factor graph. See Figure 5.1 for an example factor graph.

We define each potential function to have a log-linear form: $\psi_\alpha(\boldsymbol{y}_\alpha) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}_\alpha(\boldsymbol{y}_\alpha, \boldsymbol{x}))$. Here $\boldsymbol{x}$ is the assignment to the observed variables such as the sentence and its POS tags; $\boldsymbol{f}_\alpha$ extracts a vector of features; and $\boldsymbol{\theta}$ is our vector of model parameters. We write the resulting probability distribution over parses as $p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$, to indicate that it depends on $\boldsymbol{\theta}$.

Loss For dependency parsing, our loss function is the number of missing edges in the predicted parse $\hat{\boldsymbol{y}}$, relative to the reference (or "gold") parse $\boldsymbol{y}^*$:

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}^*) = \textstyle\sum_{i:\, \hat{y}_i = \text{OFF}} \mathbb{I}(y_i^* = \text{ON}) \tag{5.1}$$

$\mathbb{I}$ is the indicator function. Because $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}^*$ each specify exactly one parent per word token, $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}^*)$ equals the directed dependency error: the number of word tokens whose parent is predicted incorrectly.

Decoder To obtain a single parse as output, we use a minimum Bayes risk (MBR) decoder (Section 2.3.2 contained a more general discussion of MBR decoding), which returns the tree with minimum expected loss under the model's distribution (Bickel and Doksum, 1977; Goodman, 1996). Our ℓ gives the decision rule:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\operatorname{argmin}}\ \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})] \tag{5.2}$$

$$= \underset{\hat{\boldsymbol{y}}}{\operatorname{argmax}} \sum_{i:\, \hat{y}_i = \text{ON}} p_{\boldsymbol{\theta}}(y_i = \text{ON} \mid \boldsymbol{x}) \tag{5.3}$$

Here $\hat{\boldsymbol{y}}$ ranges over well-formed parses. Thus, our parser seeks a well-formed parse $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ whose individual edges have a high *probability* of being correct according to $p_{\boldsymbol{\theta}}$ (since it lacks knowledge $\boldsymbol{y}^*$ of which edges are *truly* correct). MBR is the principled way to take a loss function into account under a probabilistic model. By contrast, maximum *a posteriori* (MAP) decoding does not consider the loss function. It would return the single highest-probability parse even if that parse, and its individual edges, were unlikely to be correct.[4]

All systems in this chapter use MBR decoding to consider the loss function at *test* time. This implies that the ideal training procedure would be to find the *true* $p_{\boldsymbol{\theta}}$ so that its marginals can be used in (5.3). Our baseline system attempts this. Yet in practice, we will not be able to find the true $p_{\boldsymbol{\theta}}$ (model misspecification) nor exactly compute the marginals of $p_{\boldsymbol{\theta}}$ (computational intractability). Thus, this chapter proposes a training procedure that

[4]If we used a simple 0-1 loss function within (5.2), then MBR decoding would reduce to MAP decoding.

compensates for the system's approximations, adjusting $\boldsymbol{\theta}$ to reduce the actual loss of $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ as measured at *training* time.

To find the MBR parse, we first run inference to compute the marginal probability $p_{\boldsymbol{\theta}}(y_i = \text{ON} \mid \boldsymbol{x})$ for each edge. Then we maximize (5.3) by running a first-order dependency parser with edge scores equal to those probabilities.[5] When our inference algorithm is approximate, we replace the exact marginal with its approximation—the belief from BP, given by $b_i(\text{ON})$ in (5.6) below.

Inference Loopy belief propagation (BP) (Murphy et al., 1999) computes approximations to the variable marginals $p_{\boldsymbol{\theta}}(y_i \mid \boldsymbol{x}) = \sum_{\boldsymbol{y}':y'_i=y_i} p_{\boldsymbol{\theta}}(\boldsymbol{y}' \mid \boldsymbol{x})$, as needed by (5.3), as well as the factor marginals $p_{\boldsymbol{\theta}}(\boldsymbol{y}_\alpha \mid \boldsymbol{x}) = \sum_{\boldsymbol{y}':\boldsymbol{y}'_\alpha=\boldsymbol{y}_\alpha} p_{\boldsymbol{\theta}}(\boldsymbol{y}' \mid \boldsymbol{x})$. We reiterate the key details from Section 2.3.3 for the reader's convenience. The algorithm proceeds by iteratively sending messages from variables, y_i, to factors, α:

$$m^{(t)}_{i\to\alpha}(y_i) \propto \prod_{\beta\in\mathcal{N}(i)\setminus\alpha} m^{(t-1)}_{\beta\to i}(y_i) \tag{5.4}$$

and from factors to variables:

$$m^{(t)}_{\alpha\to i}(y_i) \propto \sum_{\boldsymbol{y}_\alpha\sim y_i} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{j\in\mathcal{N}(\alpha)\setminus i} m^{(t-1)}_{j\to\alpha}(y_i) \tag{5.5}$$

where $\mathcal{N}(i)$ and $\mathcal{N}(\alpha)$ denote the neighbors of y_i and α respectively, and where $\boldsymbol{y}_\alpha \sim y_i$ is standard notation to indicate that $\boldsymbol{y}_\alpha$ ranges over all assignments to the variables participating in the factor α provided that the ith variable has value y_i. Note that the messages at time t are computed from those at time $(t-1)$. Messages at the final time t_{max} are used to compute the *beliefs* at each factor and variable:

$$b_i(y_i) \propto \prod_{\alpha\in\mathcal{N}(i)} m^{(t_{\text{max}})}_{\alpha\to i}(y_i) \tag{5.6}$$

$$b_\alpha(\boldsymbol{y}_\alpha) \propto \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{i\in\mathcal{N}(\alpha)} m^{(t_{\text{max}})}_{i\to\alpha}(y_i) \tag{5.7}$$

We assume each of the messages and beliefs given in (5.4)–(5.7) are scaled to sum-to-one. For example, b_i is normalized such that $\sum_{y_i} b_i(y_i) = 1$ and approximates the marginal distribution over y_i values. Messages continue to change indefinitely if the factor graph is cyclic, but in the limit, the messages may converge. Although the equations above update all messages in parallel, convergence is much faster if only one message is updated per timestep, in some well-chosen *serial* order.[6]

[5]Prior work (Smith and Eisner, 2008; Bansal et al., 2014) used the log-odds ratio $\log \frac{p_{\boldsymbol{\theta}}(y_i=\text{ON})}{p_{\boldsymbol{\theta}}(y_i=\text{OFF})}$ as the edge scores for decoding, but this yields a parse different from the MBR parse.

[6]Following Dreyer and Eisner (2009) footnote 22, we choose an arbitrary directed spanning tree of the factor graph rooted at the PTREE factor. We visit the nodes in topologically sorted order (from leaves to root) and update any message from the node being visited to a node that is *later* in the order. We then reverse this order and repeat, so that every message has been passed once. This constitutes one **iteration** of BP.

For the PTREE factor, the summation over variable assignments required for $m^{(t)}_{\alpha \to i}(y_i)$ in Eq. (5.5) equates to a summation over exponentially many projective parse trees. However, we can use an inside-outside variant of Eisner (1996)'s algorithm to compute this in polynomial time (we describe this as hypergraph parsing in Section 5.3). The resulting "structured BP" inference procedure—detailed by Smith and Eisner (2008) and described in Section 2.3.3.4—is exact for first-order dependency parsing. When higher-order factors are incorporated, it is approximate but remains fast, whereas exact inference would be slow.[7]

5.3 Approximation-aware Learning

We aim to find the parameters $\boldsymbol{\theta}^*$ that minimize a regularized objective function over the training sample of (sentence, parse) pairs $\{(\boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)})\}_{d=1}^{D}$.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmin}} \, \frac{1}{D} \Big(\big(\sum_{d=1}^{D} J(\boldsymbol{\theta}; \boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)}) \big) + \frac{\lambda}{2} ||\boldsymbol{\theta}||_2^2 \Big) \tag{5.8}$$

where $\lambda > 0$ is the regularization coefficient and $J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}^*)$ is a given differentiable function, possibly nonconvex. We locally minimize this objective using ℓ_2-regularized AdaGrad with Composite Mirror Descent (Duchi et al., 2011)—a variant of stochastic gradient descent that uses mini-batches, an adaptive learning rate per dimension, and sparse lazy updates from the regularizer.[8]

Objective Functions The standard choice for J is the negative conditional log-likelihood (Section 5.6). However, as in Stoyanov et al. (2011), our aim is to minimize expected loss on the true data distribution over sentence/parse pairs (X, Y):

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} \; \mathbb{E}[\ell(h_{\boldsymbol{\theta}}(X), Y)] \tag{5.9}$$

Since the true data distribution is unknown, we substitute the expected loss over the training sample, and regularize our objective in order to reduce sampling variance. Specifically, we aim to minimize the **regularized empirical risk**, given by (5.8) with $J(\boldsymbol{\theta}; \boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)})$ set to $\ell(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(d)}), \boldsymbol{y}^{(d)})$. Note that this loss function would not be differentiable—a key issue we will take up below. This is the "ERMA" method of Stoyanov and Eisner (2012). We will also consider simpler choices of J—akin to the loss functions used by Domke (2011).

[7] How slow is exact inference for dependency parsing? For certain choices of higher-order factors, polynomial time is possible via dynamic programming (McDonald et al., 2005; Carreras, 2007; Koo and Collins, 2010). However, BP will typically be asymptotically faster (for a fixed number of iterations) and faster in practice. In some other settings, exact inference is NP-hard. In particular, non-projective parsing becomes NP-hard with even second-order factors (McDonald and Pereira, 2006). BP can handle this case in polynomial time by replacing the PTREE factor with a TREE factor that allows edges to cross.

[8] $\boldsymbol{\theta}$ is initialized to $\mathbf{0}$ when not otherwise specified.

Gradient Computation To compute the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}^*)$ of the loss on a single sentence $(\boldsymbol{x}, \boldsymbol{y}^*) = (\boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)})$, we apply automatic differentiation (AD) in the reverse mode (*Automatic Differentiation of Algorithms: Theory, Implementation, and Application* 1991). This yields the same type of "back-propagation" algorithm that has long been used for training neural networks (Rumelhart et al., 1986). It is important to note that the resulting gradient computation algorithm is exact up to floating-point error, and has the same asymptotic complexity as the original decoding algorithm, requiring only about twice the computation. The AD method applies provided that the original function is indeed differentiable with respect to $\boldsymbol{\theta}$. In principle, it is possible to compute the gradient with minimal additional coding. There exists AD software (some listed at `autodiff.org`) that could be used to derive the necessary code automatically. Another option would be to use the perturbation method of Domke (2010). However, we implemented the gradient computation directly, and we describe it here.

Inference, Decoding, and Loss as a Feedforward Circuit The backpropagation algorithm is often applied to neural networks, where the topology of a feedforward circuit is statically specified and can be applied to any input. Our BP algorithm, decoder, and loss function similarly define a feedforward circuit that computes our function J. The circuit's depth depends on the number of BP timesteps, $t_{\max}$. Its topology is defined *dynamically* (per sentence $\boldsymbol{x}^{(d)}$) by "unrolling" the computation into a graph.

Figure 5.2 shows this topology. The high level modules consist of (A) computing potential functions, (B) initializing messages, (C) sending messages, (D) computing beliefs, and (E) decoding and computing the loss. We *zoom in* on two submodules: the first computes messages from the PTREE factor efficiently (C.1–C.3); the second computes a softened version of our loss function (E.1–E.3). Both of these submodules are made efficient by the inside-outside algorithm.

The next two sections describe in greater detail how we define the function J (the forward pass) and how we compute its gradient (the backward pass). Backpropagation through the circuit from Figure 5.2 poses several challenges. Eaton and Ghahramani (2009), Stoyanov et al. (2011), and Domke (2011) showed how to backpropagate through the basic BP algorithm, and we reiterate the key details below (Section 5.5.2). The remaining challenges form the primary technical contribution of this chapter:

1. Our true loss function $\ell(h_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}^*)$ by way of the decoder $h_{\boldsymbol{\theta}}$ contains an argmax (5.3) over trees and is therefore not differentiable. We show how to soften this decoder (by substituting a softmax), making it differentiable (Section 5.4.1).

2. Empirically, we find the above objective difficult to optimize. To address this, we substitute a simpler L_2 loss function (commonly used in neural networks). This is easier to optimize and yields our best parsers in practice (Section 5.4.2).

3. We show how to run backprop through the inside-outside algorithm on a hypergraph (Section 5.5.4) for use in two modules: the softened decoder (Section 5.5.1) and computation of messages from the PTREE factor (Section 5.5.3). This allows us to go beyond Stoyanov et al. (2011) and train *structured* BP in an approximation-aware and loss-aware fashion.

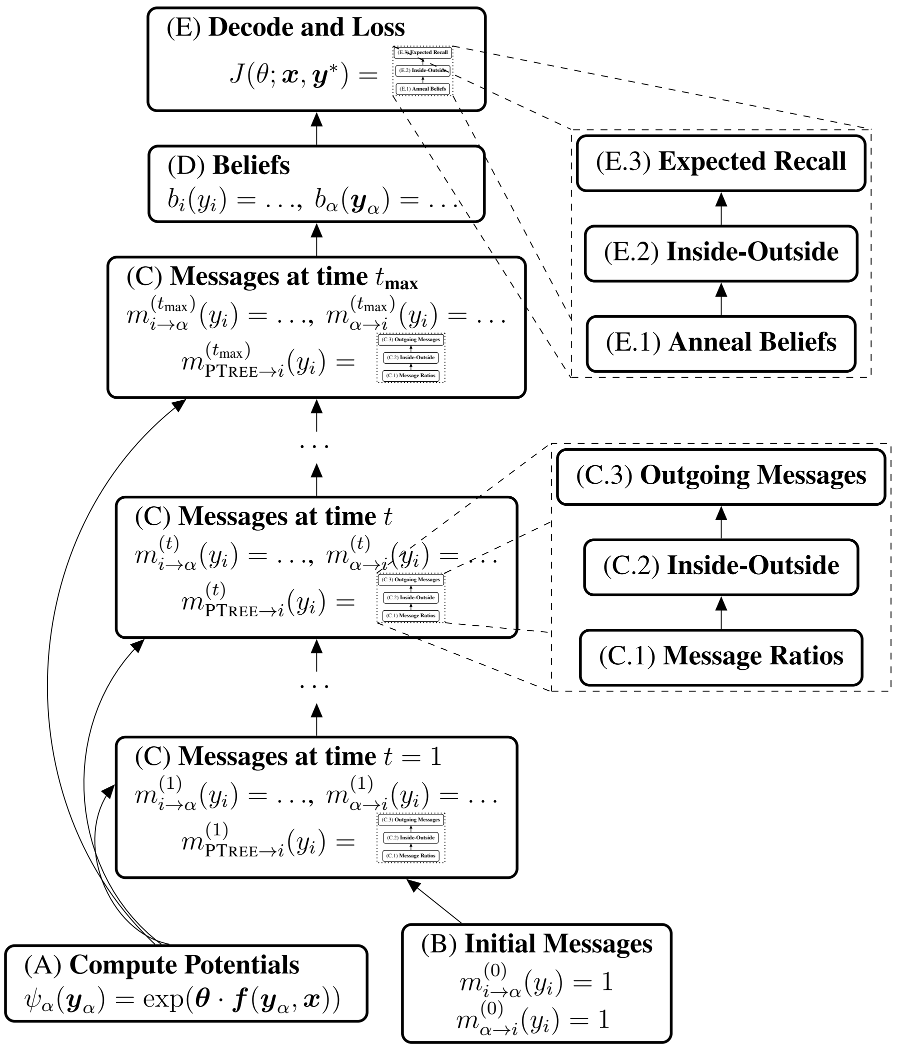

(E) Decode and Loss
$J(\theta; \boldsymbol{x}, \boldsymbol{y}^*) =$
(E.3) Expected Recall
(E.2) Inside-Outside
(E.1) Anneal Beliefs
(D) Beliefs
$b_i(y_i) = \ldots, b_\alpha(\boldsymbol{y}_\alpha) = \ldots$
(C) Messages at time $t_{\max}$
$m^{(t_{\max})}_{i\to\alpha}(y_i) = \ldots, m^{(t_{\max})}_{\alpha\to i}(y_i) = \ldots$
$m^{(t_{\max})}_{\text{PTREE}\to i}(y_i) =$
(C) Messages at time t
$m^{(t)}_{i\to\alpha}(y_i) = \ldots, m^{(t)}_{\alpha\to i}(y_i) = \ldots$
$m^{(t)}_{\text{PTREE}\to i}(y_i) =$
(C.3) Outgoing Messages
(C.2) Inside-Outside
(C.1) Message Ratios
(C) Messages at time $t = 1$
$m^{(1)}_{i\to\alpha}(y_i) = \ldots, m^{(1)}_{\alpha\to i}(y_i) = \ldots$
$m^{(1)}_{\text{PTREE}\to i}(y_i) =$
(A) Compute Potentials
$\psi_\alpha(\boldsymbol{y}_\alpha) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{y}_\alpha, \boldsymbol{x}))$
(B) Initial Messages
$m^{(0)}_{i\to\alpha}(y_i) = 1$
$m^{(0)}_{\alpha\to i}(y_i) = 1$

Figure 5.2: Feed-forward topology of inference, decoding, and loss. (E.1–E.3) show the *annealed risk*, one of the objective functions we consider.

5.4 Differentiable Objective Functions

5.4.1 Annealed Risk

Minimizing the test-time loss is the *appropriate* goal for training an approximate system like ours. That loss is estimated by the empirical risk on a large amount of in-domain supervised training data.

Alas, this risk is *nonconvex* and *piecewise constant*, so we turn to deterministic annealing (Smith and Eisner, 2006) and clever initialization. Directed dependency error, $\ell(h_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}^*)$, is not differentiable due to the argmax in the decoder $h_{\boldsymbol{\theta}}$. So we redefine $J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}^*)$ to be a new *differentiable* loss function, the **annealed risk** $R^{1/T}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}^*)$, which approaches the loss $\ell(h_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}^*)$ as the **temperature** $T \to 0$. Our first step is to define a distribution over parses, which takes the marginals $p_{\boldsymbol{\theta}}(y_i = \text{ON} \mid \boldsymbol{x})$ as input, or in practice, their BP approximations $b_i(\text{ON})$:

$$q_{\boldsymbol{\theta}}^{1/T}(\hat{\boldsymbol{y}} \mid \boldsymbol{x}) \propto \exp\left(\textstyle\sum_{i:\hat{y}_i=\text{ON}} \frac{p_{\boldsymbol{\theta}}(y_i=\text{ON}|\boldsymbol{x})}{T}\right) \tag{5.10}$$

Using this distribution, we can replace our non-differentiable decoder $h_{\boldsymbol{\theta}}$ with a differentiable one (at training time). Imagine that our new decoder stochastically returns a parse $\hat{\boldsymbol{y}}$ *sampled* from this distribution. We define the annealed risk as the expected loss of that decoder:

$$R_{\boldsymbol{\theta}}^{1/T}(\boldsymbol{x}, \boldsymbol{y}^*) = \mathbb{E}_{\hat{\boldsymbol{y}} \sim q_{\boldsymbol{\theta}}^{1/T}(\cdot|\boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}^*)] \tag{5.11}$$

As $T \to 0$ ("annealing"), the decoder almost always chooses the MBR parse,[9] so our risk approaches the loss of the actual MBR decoder that will be used at test time. However, as a function of $\boldsymbol{\theta}$, it remains differentiable (though not convex) for any $T > 0$.

To *compute* the annealed risk, observe that it simplifies to $R_{\boldsymbol{\theta}}^{1/T}(\boldsymbol{x}, \boldsymbol{y}^*) = -\sum_{i:y_i^*=\text{ON}} q_{\boldsymbol{\theta}}^{1/T}(\hat{y}_i = \text{ON} \mid \boldsymbol{x})$. This is the negated **expected recall** of a parse $\hat{\boldsymbol{y}} \sim q_{\boldsymbol{\theta}}^{1/T}$. We obtain the required marginals $q_{\boldsymbol{\theta}}^{1/T}(\hat{y}_i = \text{ON} \mid \boldsymbol{x})$ from (5.10) by running inside-outside where the edge weight for edge i is given by $\exp(p_{\boldsymbol{\theta}}(y_i = \text{ON} \mid \boldsymbol{x})/T)$.

Whether our test-time system computes the marginals of $p_{\boldsymbol{\theta}}$ exactly or does so approximately via BP, our new training objective approaches (as $T \to 0$) the true empirical risk of the test-time parser that performs MBR decoding from the computed marginals. Empirically, however, we will find that it is not the most effective training objective (Section 5.7.2). Stoyanov et al. (2011) postulate that the nonconvexity of empirical risk may make it a difficult function to optimize, even with annealing. Our next two objectives provide alternatives.

5.4.2 $\mathbf{L_2}$ Distance

We can view our inference, decoder, and loss as defining a form of deep neural network, whose topology is inspired by our linguistic knowledge of the problem (e.g., the edge variables should define a tree). This connection to deep learning allows us to consider training methods akin to supervised layer-wise training (Bengio et al., 2007). We temporarily remove the top layers of our network (i.e. the decoder and loss module, Fig. 5.2 (E)) so that the output layer of our "deep network" consists of the variable beliefs $b_i(y_i)$ from BP. We can then define a supervised loss function directly on these beliefs. We don't have supervised data for this layer of beliefs, but we can create it artificially. Use the supervised parse $\boldsymbol{y}^*$ to define "target beliefs" by $b_i^*(y_i) = \mathbb{I}(y_i = y_i^*) \in \{0, 1\}$. To find parameters $\boldsymbol{\theta}$ that make BP's beliefs close to these targets, we can minimize an **L$_2$ distance** loss function:

$$J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}^*) = \sum_i \sum_{y_i} (b_i(y_i) - b_i^*(y_i))^2 \tag{5.12}$$

We can use this L$_2$ distance objective function for training, adding the MBR decoder and loss evaluation back in only at test time.

[9]Recall from (5.3) that the MBR parse is the tree $\hat{\boldsymbol{y}}$ that maximizes the sum $\sum_{i:\hat{y}_i=\text{ON}} p_{\boldsymbol{\theta}}(y_i = \text{ON} \mid \boldsymbol{x})$. As $T \to 0$, the right-hand side of (5.10) grows fastest for this $\hat{\boldsymbol{y}}$, so its probability under $q_{\boldsymbol{\theta}}^{1/T}$ approaches 1 (or $1/k$ if there is a k-way tie for MBR parse).

5.4.3 Layer-wise Training

Just as in layer-wise training of neural networks, we can take a two-stage approach to training. First, we train to minimize the L_2 distance. Then, we use the resulting $\boldsymbol{\theta}$ as initialization to optimize the annealed risk, which does consider the decoder and loss function (i.e. the top layers of Fig. 5.2). Stoyanov et al. (2011) found mean squared error (MSE) to give a smoother training objective, though still nonconvex, and used it to initialize empirical risk. Though their variant of the L_2 objective did not completely dispense with the decoder as ours does, it is a similar approach to our proposed layer-wise training.

5.4.4 Bethe Likelihood

A key focus of this work is differentiating our method from traditional CLL training. However, it is also possible to define an objective which obtains CLL training as a special case when inference is exact. We call this objective the **Bethe likelihood** since we obtain it by replacing the true value of the log-partition function with its approximation given by the Bethe free energy. Since we do not consider this objective function in our experiments, we defer details about it to the appendix (Appendix B).

5.5 Gradients by Backpropagation

Backpropagation computes the derivative of any given function specified by an arbitrary circuit consisting of elementary differentiable operations (e.g. $+, -, \times, \div, \log, \exp$). This is accomplished by repeated application of the chain rule. Backpropagating through an *algorithm* proceeds by similar application of the chain rule, where the intermediate quantities are determined by the topology of the circuit—just as in Figure 5.2. Running backwards through the circuit, backprop computes the partial derivatives of the objective $J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}^*)$ with respect to each intermediate quantity u—or more concisely the **adjoint** of u: $\eth u = \frac{\partial J(\boldsymbol{\theta};\boldsymbol{x},\boldsymbol{y}^*)}{\partial u}$. This section describes the adjoint computations we require. Section 2.2.2 also showed additional examples of its use.

5.5.1 Backpropagation of Decoder / Loss

The adjoint of the objective itself $\eth J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}^*)$ is always 1. So the first adjoints we must compute are those of the beliefs: $\eth b_i(y_i)$ and $\eth b_\alpha(\boldsymbol{y}_\alpha)$. This corresponds to the backward pass through Figure 5.2 (E). Consider the simple case where J is L_2 *distance* from (5.12): the variable belief adjoint is $\eth b_i(y_i) = 2(b_i(y_i) - b_i^*(y_i))$ and trivially $\eth b_\alpha(\boldsymbol{y}_\alpha) = 0$. If J is *annealed risk* from (5.11), we compute $\eth b_i(y_i)$ by applying backpropagation recursively to our algorithm for J from Section 5.4.1. This sub-algorithm defines a sub-circuit depicted in Figure 5.2 (E.1–E.3). The computations of the annealed beliefs and the expected recall are easily differentiable. The main challenge is differentiating the function computed by the inside-outside algorithm; we address this in Section 5.5.4.

5.5.2 Backpropagation through Structured BP

Given the adjoints of the beliefs, we next backpropagate through *structured* BP—extending prior work which did the same for regular BP (Eaton and Ghahramani, 2009; Stoyanov et al., 2011; Domke, 2011). Except for the messages sent from the PTREE factor, each step of BP computes some value from earlier values using the update equations (5.4)–(5.7). Backpropagation differentiates these elementary expressions. First, using the belief adjoints, we compute the adjoints of the final messages ($\eth m_{j\to\alpha}^{(t_{\max})}(y_j)$, $\eth m_{\beta\to i}^{(t_{\max})}(y_i)$) by applying the chain rule to Eqs. (5.6) and (5.7). This is the backward pass through Fig. 5.2 (D). Recall that the messages at time t were computed from messages at time $t-1$ and the potential functions ψ_α in the forward pass via Eqs. (5.4) and (5.5). Backprop works in the opposite order, updating the adjoints of the messages at time $t-1$ and the potential functions ($\eth m_{j\to\alpha}^{(t-1)}(y_j)$, $\eth m_{\beta\to i}^{(t-1)}(y_i)$, $\eth\psi_\alpha(\boldsymbol{y}_\alpha)$) only *after* it has computed the adjoints of the messages at time t. Repeating this through timesteps $\{t, t-1, \ldots, 1\}$ constitutes the backward pass through Fig. 5.2 (C). The backward pass through Fig. 5.2 (B) does nothing, since the messages were initialized to a constant. The final step of backprop uses $\eth\psi_\alpha(\boldsymbol{y}_\alpha)$ to compute $\eth\theta_j$—the backward pass through Fig. 5.2 (A).

For the explicit formula of these adjoints, see Table 5.1, which provides a more complete illustration of the larger context of our backpropagation implementation. The equations are identical to those given in the appendix of Stoyanov et al. (2011), except that they are slightly modified to accommodate the notation of this thesis. The next section handles the special case of $\eth m_{j\to\text{PTREE}}^{(t)}(y_j)$.

5.5.3 BP and Backpropagation with PTREE

The PTREE factor has a special structure that we exploit for efficiency during BP. Smith and Eisner (2008) give a more efficient way to implement Eq. (5.5), which computes the message from a factor α to a variable y_i, in the special case where $\alpha =$ PTREE. They first run the inside-outside algorithm where the edge weights are given by the ratios of the messages to PTREE: $\frac{m_{i\to\alpha}^{(t)}(\text{ON})}{m_{i\to\alpha}^{(t)}(\text{OFF})}$. Then they multiply each resulting edge marginal given by inside-outside by the product of all the OFF messages $\prod_i m_{i\to\alpha}^{(t)}(\text{OFF})$ to get the marginal factor belief $b_\alpha(y_i)$. Finally they divide the belief by the incoming message $m_{i\to\alpha}^{(t)}(\text{ON})$ to get the corresponding outgoing message $m_{\alpha\to i}^{(t+1)}(\text{ON})$. These steps are shown in Figure 5.2 (C.1–C.3), and are repeated each time we send a message from the PTree factor. (We provide additional details in Section 2.3.3.4.)

Similarly, we exploit the structure of this algorithm to compute the adjoints $\eth m_{j\to\text{PTREE}}^{(t)}(y_j)$. The derivatives of the message ratios and products mentioned here are simple. In the next subsection, we explain how to backpropagate through the inside-outside algorithm. Though we focus here on projective dependency parsing, our techniques are also applicable to non-projective parsing and the TREE factor; we leave this to future work.

Category	Forward Computation	Backward Computation
Belief: Variable	$\tilde{b}_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} m^{(T)}_{\alpha \to i}(x_i)$	$\eth m^{(T)}_{\beta \to i}(x_i) = \eth \tilde{b}_i(x_i) \prod_{\alpha \in \mathcal{N}(i) \setminus \beta} m^{(T)}_{\alpha \to i}(x_i)$
Belief: Factor	$\tilde{b}_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{i \in \mathcal{N}(\alpha)} m^{(T)}_{i \to \alpha}(x_i)$	$\eth m^{(T)}_{j \to \alpha}(x_j) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}} \sim x_j} \eth \tilde{b}_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{i \in \mathcal{N}(\alpha) \setminus j} m^{(T)}_{i \to \alpha}(x_i)$ $\eth \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \eth b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j \in \mathcal{N}(\alpha)} m^{(T)}_{j \to \alpha}(x_j)$
Message: Variable to Factor	$\tilde{m}^{(t)}_{i \to \alpha}(x_i) = \prod_{\beta \in \mathcal{N}(i) \setminus \alpha} m^{(t-1)}_{\beta \to i}(x_i)$	$\eth m^{(t-1)}_{\beta \to i}(x_i) \mathrel{+}= \eth \tilde{m}^{(t)}_{i \to \alpha}(x_i) \prod_{\gamma \in \mathcal{N}(i) \setminus \{\alpha, \beta\}} m^{(t-1)}_{\gamma \to i}(x_i)$
Message: Factor to Variable	$\tilde{m}^{(t)}_{\alpha \to i}(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}} \sim x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} m^{(t-1)}_{j \to \alpha}(x_i)$	$\eth \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \mathrel{+}= \eth \tilde{m}^{(t)}_{\alpha \to i}(x_i) \prod_{j \in \mathcal{N}(\alpha) \setminus i} m^{(t-1)}_{j \to \alpha}(x_j)$ $\eth m^{(t-1)}_{j \to \alpha}(x_j) \mathrel{+}= \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}} \sim x_j} \eth \tilde{m}^{(t)}_{\alpha \to i}(x_i) \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{k \in \mathcal{N}(\alpha) \setminus \{i,j\}} m^{(t-1)}_{k \to \alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[k])$
Normalize	$q(x_i) = \frac{\tilde{q}(x_i)}{\sum_{x_j} \tilde{q}(x_j)}$	$\eth \tilde{q}(x_i) = \frac{1}{\sum_{x_j} \tilde{q}(x_j)} \left(\eth q(x_i) - \sum_{x_j} \eth q(x_j) q(x_j) \right)$
Potentials	$\psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_{\boldsymbol{\alpha}})) = \exp\left(\sum_j \theta_j f_j(\boldsymbol{x}_{\boldsymbol{\alpha}}) \right)$	$\eth \theta_j \mathrel{+}= \sum_\alpha \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}} \eth \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) f_j(\boldsymbol{x}_{\boldsymbol{\alpha}})$
Initial Message: Variable to Factor	$m^{(0)}_{i \to \alpha}(x_i) = 1$	
Initial Message: Factor to Variable	$m^{(0)}_{\alpha \to i}(x_i) = 1$	

Table 5.1: Belief propagation unrolled through time. Here we show the forward computation, and the backward computation used to compute derivatives. These equations can also be found in the appendix of (Stoyanov et al., 2011), but we include them here for reference.

5.5.4 Backprop of Hypergraph Inside-Outside

Both the annealed risk loss function (Section 5.4.1) and the computation of messages from the PTREE factor (Section 5.5.3) use the inside-outside algorithm for dependency parsing. Here we describe inside-outside and the accompanying backpropagation algorithm over a *hypergraph*. This general treatment (Klein and Manning, 2001; Li and Eisner, 2009) enables our method to be applied to other tasks such as constituency parsing, HMM forward-backward, and hierarchical machine translation. In the case of dependency parsing, the structure of the hypergraph is given by the dynamic programming algorithm of Eisner (1996).

For the **forward pass** of the inside-outside module, the input variables are the hyperedge weights $w_e \forall e$ and the outputs are the marginal probabilities $p_w(i) \forall i$ of each node i in the hypergraph. The latter are a function of the inside β_i and outside α_j probabilities. We initialize $\alpha_{\text{root}} = 1$.

$$\beta_i = \sum_{e \in I(i)} w_e \prod_{j \in T(e)} \beta_j \tag{5.13}$$

$$\alpha_j = \sum_{e \in O(i)} w_e \, \alpha_{H(e)} \prod_{j \in T(e): j \neq i} \beta_j \tag{5.14}$$

$$p_w(i) = \alpha_i \beta_i / \beta_{\text{root}} \tag{5.15}$$

For each node i, we define the set of incoming edges $I(i)$ and outgoing edges $O(i)$. The antecedents of the edge are $T(e)$, the parent of the edge is $H(e)$, and its weight is w_e.

For the **backward pass** of the inside-outside module, the inputs are $\eth p_w(i) \forall i$ and the outputs are $\eth w_e \forall e$. We also compute the adjoints of the intermediate quantities $\eth \beta_j, \eth \alpha_i$. We first compute $\eth \alpha_i$ bottom-up. Next $\eth \beta_j$ are computed top-down. The adjoints $\eth w_e$ are then computed in any order.

$$\eth \alpha_i = \eth p_w(i) \frac{\partial p_w(i)}{\partial \alpha_i} + \sum_{e \in I(i)} \sum_{j \in T(e)} \eth \alpha_j \frac{\partial \alpha_j}{\partial \alpha_i} \tag{5.16}$$

$$\eth \beta_{\text{root}} = \sum_{i \neq \text{root}} \eth p_w(i) \frac{\partial p_w(i)}{\partial \beta_{\text{root}}} \tag{5.17}$$

$$\begin{aligned} \eth \beta_j = {} & \eth p_w(j) \frac{\partial p_w(j)}{\partial \beta_j} + \sum_{e \in O(j)} \eth \beta_{H(e)} \frac{\partial \beta_{H(e)}}{\partial \beta_j} \\ & + \sum_{e \in O(j)} \sum_{k \in T(e): k \neq j} \eth \alpha_k \frac{\partial \alpha_k}{\partial \beta_j} \quad \forall j \neq \text{root} \end{aligned} \tag{5.18}$$

$$\eth w_e = \eth \beta_{H(e)} \frac{\partial \beta_{H(e)}}{\partial w_e} + \sum_{j \in T(e)} \eth \alpha_j \frac{\partial \alpha_j}{\partial w_e} \tag{5.19}$$

Below, we show the partial derivatives required for the adjoint computations in Section 5.5.4.

$$\frac{\partial p_w(i)}{\partial \alpha_i} = \beta_i / \beta_{\text{root}}, \qquad \frac{\partial p_w(i)}{\partial \beta_{\text{root}}} = -\alpha_i \beta_i / (\beta_{\text{root}}^2),$$
$$\frac{\partial p_w(i)}{\partial \beta_i} = \alpha_i / \beta_{\text{root}}$$

For some edge e, let $i = H(e)$ be the parent of the edge and $j, k \in T(e)$ be among its antecedents.

$$\frac{\partial \beta_i}{\partial \beta_j} = w_e \prod_{k \in T(e): k \neq j} \beta_k, \qquad \frac{\partial \beta_{H(e)}}{\partial w_e} = \prod_{j \in T(e)} \beta_j$$
$$\frac{\partial \alpha_j}{\partial \alpha_i} = w_e \prod_{k \in T(e): k \neq j} \beta_k, \qquad \frac{\partial \alpha_j}{\partial w_e} = \alpha_{H(e)} \prod_{k \in T(e): k \neq j} \beta_k$$
$$\frac{\partial \alpha_k}{\partial \beta_j} = w_e \alpha_H(e) \prod_{l \in T(e): l \neq j, l \neq k} \beta_l$$

This backpropagation method is used for both Figure 5.2 (C.2) and (E.2).

5.6 Other Learning Settings

Loss-aware Training with Exact Inference Backpropagating through inference, decoder, and loss need not be restricted to *approximate* inference algorithms. Li and Eisner (2009) optimize Bayes risk with exact inference on a hypergraph for machine translation. Each of our differentiable loss functions (Section 5.4) can also be coupled with exact inference. For a first-order parser, BP is exact. Yet, in place of modules (B), (C), and (D) in Figure 5.2, we can use a standard dynamic programming algorithm for dependency parsing, which is simply another instance of inside-outside on a hypergraph (Section 5.5.4). The exact marginals from inside-outside (5.15) are then fed forward into the decoder/loss module (E).

Conditional and Surrogate Log-likelihood The standard approach to training is conditional log-likelihood (CLL) maximization (Smith and Eisner, 2008) without taking inexact inference into account: $J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}^*) = -\log p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$. The gradient is computed by hand as the difference between observed and expected feature counts. When inference is exact, this baseline computes the true gradient of CLL. When inference is approximate, this baseline uses the factor beliefs $b_\alpha(\boldsymbol{y}_\alpha)$ from BP in place of the exact marginals in the gradient. The literature refers to this approximation-*un*aware training method as *surrogate likelihood* training since it returns the "wrong" parameters even under the assumption of infinite training data drawn from the model being used (Wainwright, 2006). For BP, the exact objective it is optimizing (i.e. antiderivative of the gradient) is not known, so one must use an optimizer that doesn't require the function value (e.g. SGD). Despite this, the surrogate likelihood objective is commonly used to train CRFs. CLL and approximation-aware training are not mutually exclusive. Training a standard factor graph with ERMA and a log-likelihood objective recovers CLL exactly (Stoyanov et al., 2011).

5.7 Experiments

5.7.1 Setup

Features As the focus of this work is on a novel approach to training, we look to prior work for model and feature design (Section 5.2). We add $O(n^3)$ second-order grandparent and arbitrary-sibling factors as in Riedel and Smith (2010) and Martins et al. (2010b). We use standard feature sets for first-order (McDonald et al., 2005) and second-order (Carreras, 2007) parsing. Following Rush and Petrov (2012), we also include a version of each part-of-speech (POS) tag feature, with the coarse tags from Petrov et al. (2012). We use feature hashing (Ganchev and Dredze, 2008; Weinberger et al., 2009) and restrict to at most 20 million features. We leave the incorporation of third-order features to future work.

Pruning To reduce the time spent on feature extraction, we enforce the *type-specific* dependency length bounds from Eisner and Smith (2005) as used by Rush and Petrov (2012): the maximum allowed dependency length for each tuple (parent tag, child tag, direction) is given by the maximum observed length for that tuple in the training data. Following Koo and Collins (2010), we train a first-order model with CLL and for each token prune any parents for which the marginal probability is less than 0.0001 times the maximum parent marginal for that token. On a per-token basis, we further restrict to the ten parents with highest marginal probability as in Martins et al. (2009) (but we avoid pruning the fully right-branching tree, so that some parse always exists).[10] This lets us simplify the factor graph, removing variables y_i corresponding to pruned edges and specializing their factors to assume $y_i = \text{OFF}$. We train the full model's parameters to work well on this pruned graph.

Data and Evaluation We consider 19 languages from the CoNLL-2006 (Buchholz and Marsi, 2006) and CoNLL-2007 (Nivre et al., 2007) Shared Tasks. We also convert the English Penn Treebank (PTB) (Marcus et al., 1993) to dependencies using the head rules from Yamada and Matsumoto (2003) (PTB-YM). We evaluate unlabeled attachment accuracy (UAS) using gold POS tags for the CoNLL languages, and predicted tags from TurboTagger (Martins et al., 2013) for the PTB. Following prior work, we exclude punctuation when evaluating the English PTB data, but include punctuation for all the CoNLL datasets. Unlike most prior work, we hold out 10% of each CoNLL training dataset as development data for regularization by early stopping.[11]

Some of the CoNLL languages contain non-projective edges, but our system is built using a probability distribution over projective trees only. ERMA can still be used with such a badly misspecified model—one of its advantages—but no amount of training can raise CLL's objective above $-\infty$, since any non-projective gold tree will always have probability 0. Thus, for CLL only, we replace each gold tree in training data with a minimum-loss

[10]The pruning model uses a simpler feature set as in Rush and Petrov (2012). Pruning is likely the least impactful of our approximations: it obtains 99.46% oracle UAS for English.

[11]In dev experiments, we found L_2 distance to be less sensitive to the ℓ_2-regularizer weight than CLL. So we added additional regularization by early stopping to improve CLL.

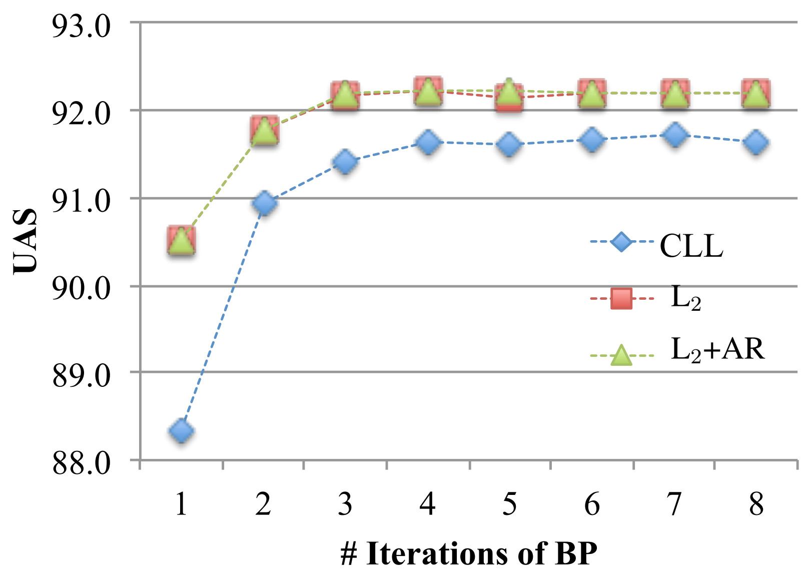


Figure 5.3: Speed/accuracy tradeoff of English PTB-YM UAS vs. the *total* number of BP iterations t_{max} for standard conditional likelihood training (CLL) and our approximation-aware training with either an L_2 objective (L_2) or a staged training of L_2 followed by annealed risk (L_2+AR). The UAS excludes punctuation. Note that the x-axis shows the number of iterations used for *both* training and testing. We use a 2nd-order model with Grand.+Sib. factors.

projective tree (Carreras, 2007).[12] This resembles ERMA's goal of training the system to find a low-loss projective tree. At test time, we always evaluate the system's projective output trees against the possibly non-projective gold trees, as in prior work.

To test the statistical significance of our results on UAS, we use the approximate randomization test (aka. paired permutation test) with 10^6 samples. We found the p-values were similar (slightly more conservative) than those given by the paired bootstrap test.

Learning Settings We compare three learning settings. The first, our baseline, is conditional log-likelihood training (CLL) (Section 5.6). As is common in the literature, we conflate two distinct learning settings (conditional log-likelihood/surrogate log-likelihood) under the single name "CLL," allowing the inference method (exact/inexact) to differentiate them. The second learning setting is approximation-aware learning (Section 5.3) with either our L_2 distance objective (L_2) (Section 5.4.2) or our layer-wise training method (L_2+AR) which takes the L_2-trained model as an initializer for our annealed risk (Section 5.4.3). The annealed risk objective requires an annealing schedule: over the course of training, we linearly anneal from initial temperature $T = 0.1$ to $T = 0.0001$, updating T at each step of stochastic optimization. The third learning setting uses the same two objectives, L_2 and L_2+AR, but with exact inference (Section 5.6). The ℓ_2-regularizer weight in (5.8) is $\lambda = 1$. Each method is trained by AdaGrad for 5 epochs with early stopping

[12]We also ran a controlled experiment with L_2 and not just CLL trained on these *projectivized* trees: the average margin of improvement for our method widened very slightly.

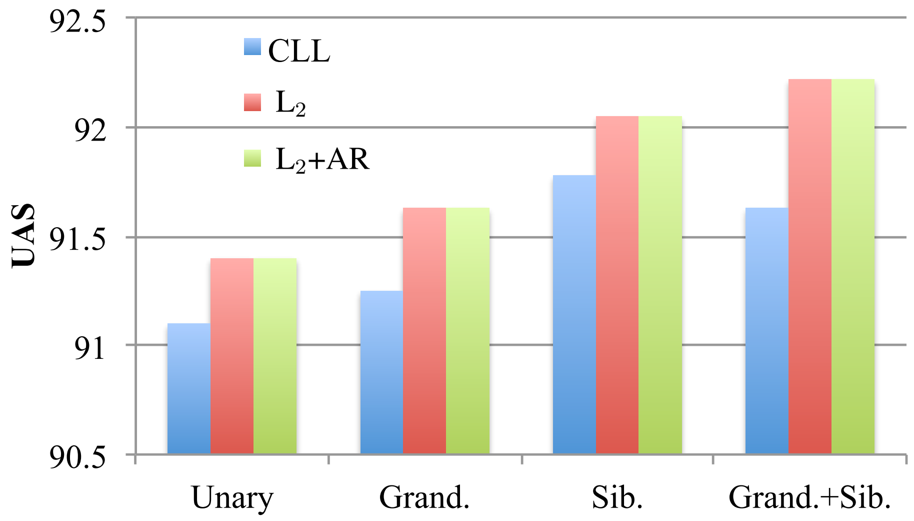


Figure 5.4: English PTB-YM UAS vs. the types of 2nd-order factors included in the model for approximation-aware training and standard conditional likelihood training. The UAS excludes punctuation. All models include 1st-order factors (Unary). The 2nd-order models include grandparents (Grand.), arbitrary siblings (Sib.), or both (Grand.+Sib.)—and use 4 iterations of BP. For each of these models, the improvement given by training with our method instead of CLL is statistically significant at the $p < 0.005$ level.

(i.e. the model with the highest score on dev data is returned). Across CoNLL, the average epoch chosen for CLL was 2.02 and for L_2 was 3.42. The learning rate for each training run is dynamically tuned on a sample of the training data.

5.7.2 Results

Our goal is to demonstrate that our approximation-aware training method leads to improved parser accuracy as compared with the standard training approach of conditional log-likelihood (CLL) maximization (Smith and Eisner, 2008), which does not take inexact inference into account. The two key findings of our experiments are that our learning approach is more robust to (1) decreasing the number of iterations of BP and (2) adding additional cycles to the factor graph in the form of higher-order factors. In short: our approach leads to faster inference and creates opportunities for more accurate parsers.

Speed-Accuracy Tradeoff Our first experiment is on English dependencies. For English PTB-YM, Figure 5.3 shows accuracy as a function of the number of BP iterations for our second-order model with both arbitrary sibling and grandparent factors on English. We find that our training methods (L_2 and L_2+AR) obtain higher accuracy than standard training (CLL), particularly when a small number of BP iterations are used and the inference is a worse approximation. Notice that with just *two* iterations of BP, the parsers trained by our approach obtain accuracy greater than or equal to those by CLL with *any* number of iterations (1 to 8). Contrasting the two objectives for our approximation-aware training, we find that our simple L_2 objective performs very well. In fact, in only two cases, at 3 and 5 iterations, does risk annealing (L_2+AR) further improve performance on test data. In our development experiments, we also evaluated AR without using L_2 for initialization and we found that it performed worse than either of CLL and L_2 alone. That AR performs only slightly better than L_2 (and not worse) in the case of L_2+AR is likely due to early stopping on dev data, which guards against selecting a worse model.

Increasingly Cyclic Models Figure 5.4 contrasts accuracy with the type of 2nd-order factors (grandparent, sibling, or both) included in the model for English, for a fixed budget of 4 BP iterations. Adding higher-order factors introduces more loops, making the loopy BP approximation more problematic for standard CLL training. By contrast, under approximation-aware training, enriching the model with more factors always *helps* performance, as desired, rather than *hurting* it.

The UAS improvements given by our training method over CLL are significant at the $p < 0.005$ level for each model we considered in Figure 5.4. The UAS for Sib. and Grand.+Sib. with CLL training are statistically indistinguishable in Figure 5.4, despite the noticeable drop. However, with approximation-aware training, the improvement from Sib. to Grand.+Sib. is significant with $p = 0.006$.

Notice that our advantage is not restricted to the case of loopy graphs. Even when we use a 1st-order model, for which BP inference is exact, our approach yields higher-accuracy parsers than CLL training. We speculate that this improvement is due to our method's ability to better deal with model misspecification—a first-order model is quite misspecified! Note the following subtle point: when inference is exact, the CLL estimator is actually a special case of our approximation-aware learner—that is, CLL computes the same gradient that our training by backpropagation would if we used log-likelihood as the objective.

Exact Inference with Grandparents Section 5.2 noted that since we always do MBR decoding, the ideal strategy is to fit the true distribution with a good model. Consider a "good model" that includes unary and grandparent factors. Exact inference is possible here in $O(n^4)$ time by dynamic programming (Koo and Collins, 2010, Model 0). Table 5.2 shows that CLL training with exact inference indeed does well on test data—but that accuracy falls if we substitute fast approximate inference (4 iterations of BP). Our proposed L_2 training is able to close the gap, just as intended. That is, we succesfully train a few iterations of an approximate $O(n^3)$ algorithm to behave as well as an exact $O(n^4)$ algorithm.

Other Languages Our final experiments train and test our parsers on 19 languages from CoNLL-2006/2007 (Table 5.3). We find that, on average across languages, approximation-aware training with an L_2 objective obtains higher UAS than CLL training. This result holds for both our poorest model (1st-order) and our richest one (2nd-order with grandparent and sibling factors), using 1, 2, 4, or 8 iterations of BP. Figure 5.5 presents the results of Table 5.3 visually. Notice that the approximation-aware training doesn't always outperform CLL training—only in the aggregate. Again, we see the trend that our training approach yields larger gains when BP is restricted to a small number of maximum iterations. It is possible

TRAIN	INFERENCE	DEV UAS	TEST UAS
CLL	Exact	91.99	91.62
CLL	BP 4 iters	91.37	91.25
L_2	Exact	91.91	91.66
L_2	BP 4 iters	91.83	91.63

Table 5.2: The impact of exact vs. approximate inference on a 2nd-order model with grandparent factors only. Results are for the development (§ 22) and test (§ 23) sections of PTB-YM.

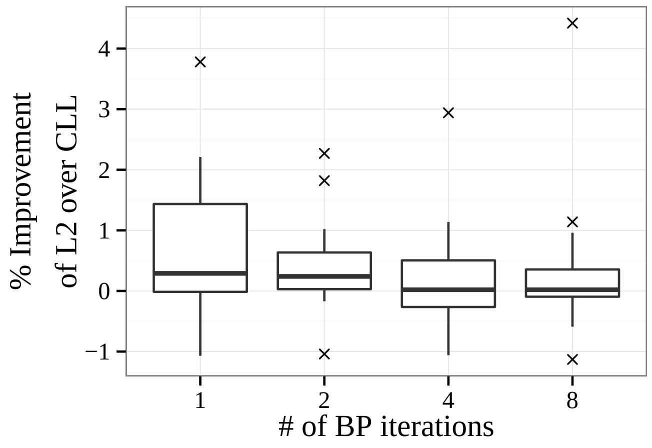


Figure 5.5: Improvement in unlabeled attachment score on test data (UAS) given by using our training method (L_2) instead of conditional log-likelihood training (CLL) for 19 languages from CoNLL-2006/2007. The improvements are calculated directly from the results in Table 5.3.

that larger training sets would also favor our approach, by providing a clearer signal of how to reduce the objective (5.8).

5.8 Discussion

The purpose of this work was to explore ERMA and related training methods for models which incorporate structured factors. We applied these methods to a basic higher-order dependency parsing model, because that was the simplest and first instance of structured BP (Smith and Eisner, 2008). In future work, we hope to explore further models with structured factors—particularly those which jointly account for multiple linguistic strata (e.g. syntax, semantics, and topic). Another natural extension of this work is to explore other types of factors: here we considered only log-linear potential functions (commonly used in CRFs), but any differentiable function would be appropriate, such as a neural network (Durrett and Klein, 2015; Gormley et al., 2015b).

Our primary contribution is approximation-aware training for structured BP. We have specifically presented message-passing formulas for any factor whose belief's partition

	1ST-ORDER		2ND-ORDER (WITH GIVEN NUM. BP ITERATIONS)							
			1		2		4		8	
LANGUAGE	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$
AR	77.63	-0.26	73.39	**+2.21**	77.05	-0.17	77.20	+0.02	77.16	-0.07
BG	90.38	**-0.76**	89.18	-0.45	90.44	+0.04	90.73	+0.25	90.63	-0.19
CA	90.47	+0.30	88.90	+0.17	90.79	+0.38	91.21	**+0.78**	91.49	**+0.66**
CS	84.69	-0.07	79.92	**+3.78**	82.08	**+2.27**	83.02	**+2.94**	81.60	**+4.42**
DA	87.15	-0.12	86.31	**-1.07**	87.41	+0.03	87.65	-0.11	87.68	-0.10
DE	88.55	**+0.81**	88.06	0.00	89.27	+0.46	89.85	-0.05	89.87	-0.07
EL	82.43	**-0.54**	80.02	+0.29	81.97	+0.09	82.49	-0.16	82.66	-0.04
EN	88.31	+0.32	85.53	**+1.44**	87.67	**+1.82**	88.63	**+1.14**	88.85	**+0.96**
ES	81.49	-0.09	79.08	-0.37	80.73	+0.14	81.75	-0.66	81.52	+0.02
EU	73.69	+0.11	71.45	**+0.85**	74.16	+0.24	74.92	-0.32	74.94	-0.38
HU	78.79	-0.52	76.46	**+1.24**	79.10	+0.03	79.07	+0.60	79.28	+0.31
IT	84.75	+0.32	84.14	+0.04	85.15	+0.01	85.66	-0.51	85.81	-0.59
JA	93.54	+0.19	93.01	**+0.44**	93.71	-0.10	93.75	-0.26	93.47	+0.07
NL	76.96	+0.53	74.23	**+2.08**	77.12	+0.53	78.03	-0.27	77.83	-0.09
PT	86.31	+0.38	85.68	-0.01	87.01	+0.29	87.34	+0.08	87.30	+0.17
SL	79.89	+0.30	78.42	**+1.50**	79.56	**+1.02**	80.91	+0.03	80.80	+0.34
SV	87.22	**+0.60**	86.14	-0.02	87.68	**+0.74**	88.01	+0.41	87.87	+0.37
TR	78.53	-0.30	77.43	**-0.64**	78.51	**-1.04**	78.80	**-1.06**	78.91	**-1.13**
ZH	84.93	-0.39	82.62	**+1.43**	84.27	**+0.95**	84.79	**+0.68**	84.77	**+1.14**
AVG.	83.98	+0.04	82.10	+0.68	83.88	+0.41	84.41	+0.19	84.34	+0.31

Table 5.3: Results on 19 languages from CoNLL-2006/2007. For languages appearing in both datasets, the 2006 version was used, except for Chinese (ZH). We report *absolute* UAS for the baseline (CLL) and the *improvement* in UAS for L_2 over CLL ($L_2 - \text{CLL}$) with positive/negative differences in blue/red. The UAS includes punctuation. The average UAS and average difference across all languages (AVG.) is given. Differences (better/worse) in **bold** are statistically significant at the $p < 0.05$ level. These results are further analyzed in Figure 5.5.

function can be computed as the total weight of all hyperpaths in a weighted hypergraph. This would suffice to train the structured BP systems that have been built for projective dependency parsing (Smith and Eisner, 2008), CNF grammar parsing (Naradowsky et al., 2012b), TAG (Auli and Lopez, 2011), ITG-constraints for phrase extraction (Burkett and Klein, 2012), and graphical models over strings (Dreyer and Eisner, 2009).

5.9 Summary

We introduce a new approximation-aware learning framework for belief propagation with structured factors. We present differentiable objectives for both empirical risk minimization (à la ERMA) and a novel objective based on L_2 distance between the inferred beliefs and the true edge indicator functions. Experiments on the English Penn Treebank and 19 languages from CoNLL-2006/2007 shows that our estimator is able to train more accurate dependency parsers with fewer iterations of belief propagation than standard conditional log-likelihood training, by taking approximations into account.

Chapter 6

Graphical Models with Structured and Neural Factors and Approximation-aware Learning

The previous chapters have illustrated three key points: (1) latent variables are an effective modeling tool that can outperform some grammar induction systems (Chapter 3), (2) both traditional hand-crafted features and learned features can be treated as factors in a factor graph (Chapter 4), and (3) for structured graphical models with cycles, approximation-aware training can yield faster and more accurate systems (Chapter 5).

In this chapter, we combine the methods from the previous three chapters in order to obtain the benefits of them all. We propose graphical models with structured factors, neural factors, and approximation-aware training in a semi-supervised setting. Following our original motivation, we focus here on a low-resource setting for semantic role labeling where a joint model with latent dependency and tagging syntax improves our overall performance.

6.1 Introduction

Many tasks in NLP focus on a single linguistic stratum when in fact we care about several. The reasons for this are often practical: machine learning does not provide the tools that allow one to readily create a joint model of multiple levels of linguistic data.

The types of models we hope to build would elegantly handle low-resource settings, taking advantage of whatever data is available. Though fully unsupervised methods (e.g. (Smith, 2006; Spitkovsky, 2013)) provide one option, they are not catered to a specific task and a small amount of supervision can often outperform them (Naseem et al., 2010; Søgaard, 2012). For the task of semantic role labeling (SRL), it is difficult to say whether joint modeling is worth the extra effort when supervised training data abounds.[1] However, in low-resource settings, the advantages of joint modeling are clearer (Boxwell et al., 2011; Naradowsky et al., 2012a; Gormley et al., 2014) (Chapter 3).

[1]The top performers in the CoNLL-2009 shared task (Gesmundo et al., 2009; Hajič et al., 2009; Lluís et al., 2013) for joint syntactic and semantic dependency parsing provide evidence of this.

Because our focus is on NLP, we seek to build models that allow declarative constraints to be specified over a set of variables. This arises in many tasks such as dependency parsing (Riedel and Clarke, 2006; Smith and Eisner, 2008; Martins et al., 2009), constituency parsing (Naradowsky et al., 2012b), phrase extraction (Burkett and Klein, 2012), TAG (Auli and Lopez, 2011), and SRL (Das et al., 2012). Dual decomposition and other techniques allow for MAP inference in these sorts of models (Duchi et al., 2006; Riedel and Clarke, 2006; Martins et al., 2009; Koo et al., 2010; Martins et al., 2011b). However, because of our interest in *low-resource* settings we expect that it will be useful to marginalize over the unobserved variables in our model—so we turn to marginal inference by structured BP (Smith and Eisner, 2008).

These *inexact* inference techniques can cause problems for standard learning algorithms (Kulesza and Pereira, 2008). For MAP inference there exist algorithms that can handle this inexact inference (Huang et al., 2012; Zhang et al., 2013). But for marginal inference the existing algorithms can't handle structured factors (Stoyanov et al., 2011; Domke, 2011).

Finally, a variety of work old (Bengio et al., 1990; Bengio et al., 1992; Haffner, 1993; Bengio and Frasconi, 1995; Bengio et al., 1995; Bourlard et al., 1995) and new (Ning et al., 2005; Morin and Bengio, 2005; Do and Artieres, 2010; Tompson et al., 2014; Srikumar and Manning, 2014) has explored hybrids of graphical models and neural networks for structured prediction. Applications of these techniques have included SRL (Collobert and Weston, 2008; Foland and Martin, 2015; FitzGerald et al., 2015). However, none of this work handles the case of structured factors, latent variables, neural factors, and inexact inference that we are concerned with here.

In this chapter we introduce a framework that permits (a) structural constraints over latent variables, (b) learned features, (c) efficient approximate inference, and (d) learning that performs well despite any approximations made by our system. We demonstrate its effectiveness on the task of low-resource semantic role labeling. The introduction of this framework is at the core of the contributions of this chapter:

- We introduce a new variety of hybrid graphical models and neural networks.
- We propose approximation-aware training for structured belief propagation with neural factors.
- We unify three forms of inference: BP on factor graphs, inside-outside on a hypergraph, and feed-forward computation in a neural network.
- We introduce a joint model of this type for semantic role labeling, syntactic dependency parsing, and part-of-speech tagging.
- We study this model in a low-resource setting for SRL that treats the syntax (parse and tags) as latent and trains in a semi-supervised fashion.

We begin by introducing a novel graphical model with structured and neural factors (Section 6.2). Taking a probabilistic perspective of the model, we describe how to carry out approximate inference (Section 6.3), decoding (Section 6.4), and approximation-*un*aware surrogate likelihood training (Section 6.5.1). Finally, we train the same system to be approximation-*aware* (Section 6.5.2). Doing so leads to an alternative perspective of our model as a deep neural network whose topology is inspired by approximate inference on the graphical model of this section.

6.2 Model

We introduce a model for joint semantic role labeling, syntactic dependency parsing, and part-of-speech tagging. Note however that we will use this model in a semi-supervised setting: during training, we will observe semantic roles for each sentence, but not syntactic dependencies or part-of-speech tags—the same low-resource setting as Chapter 3. Accordingly, the syntax will be treated as latent and will only act in the service of our semantic role labeler.

Semantic Role Labeler Our semantic role labeler is a conditional model $p_{\boldsymbol{\theta}}(\boldsymbol{r} \mid \boldsymbol{x})$, which is defined by marginalizing our joint model for syntax and semantics $p_{\boldsymbol{\theta}}(\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{t} \mid \boldsymbol{x})$. For the conditional model, the input $\boldsymbol{x}$ is a sentence. An output assignment $\boldsymbol{r}$ encodes a semantic role labeling of the sentence. The latent structure $\{\boldsymbol{e}, \boldsymbol{t}\}$ consists of a syntactic dependency tree $\boldsymbol{e}$ and a part-of-speech tagging $\boldsymbol{t}$. The probability of a semantic role labeling $\boldsymbol{r}$ for a given sentence $\boldsymbol{x}$ can thus be written in form:

$$p_{\boldsymbol{\theta}}(\boldsymbol{r} \mid \boldsymbol{x}) = \sum_{\boldsymbol{e},\boldsymbol{t}} p_{\boldsymbol{\theta}}(\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{t} \mid \boldsymbol{x}) \tag{6.1}$$

This distribution defines the probability of the output variables $\boldsymbol{R}$ given the input variables $\boldsymbol{X}$, *marginalizing* over the latent variables $\{\boldsymbol{E}, \boldsymbol{T}\}$. The form of the joint model $p_{\boldsymbol{\theta}}(\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{t} \mid \boldsymbol{x})$ is discussed below.

Joint Model of Syntax and Semantics Our joint model $p_{\boldsymbol{\theta}}(\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{t} \mid \boldsymbol{x})$ defines the probability of the semantics $\boldsymbol{r}$ *and* latent syntax $\{\boldsymbol{e}, \boldsymbol{t}\}$ given the sentence $\boldsymbol{x}$. We will describe this *joint* model as a factor graph (Frey et al., 1997; Kschischang et al., 2001). We follow our definition of factor graphs given in Section 2.3.1. For conciseness, we abbreviate the full set of output variables for the joint model as $\boldsymbol{Y} = \{\boldsymbol{R}, \boldsymbol{E}, \boldsymbol{T}\}$. The probability is proportional to a product of non-negative potential functions ψ_α:

$$p_{\boldsymbol{\theta}}(\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{t} \mid \boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{\alpha} \psi_\alpha(\boldsymbol{y}_\alpha \boldsymbol{x}) \tag{6.2}$$

where $Z(\boldsymbol{x})$ is the sentence-specific partition function ensuring that the distribution sums to one. One of the main contributions of this chapter is that the potential functions (which are in one-to-one correspondence with factors α) come in one of three forms:

Log-linear factors These constitute the standard potential function for a conditional random field (CRF) (Lafferty et al., 2001) having the form $\psi_\alpha(\boldsymbol{y}_\alpha) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}_\alpha(\boldsymbol{y}_\alpha, \boldsymbol{x}))$. In our model, we define a log-linear factor for each variable. However, we also include factors over *pairs* of variables. These connect the dependency edge variables $\boldsymbol{E}$ to the roles $\boldsymbol{R}$, and the tag variables $\boldsymbol{T}$ to the roles $\boldsymbol{R}$.

Neural factors Potential functions for these factors are defined by the score of an FCM neural network from Section 4.3. While these neural factors would be appropriate for all the variables, we only include them as unary factors on the semantic roles $\boldsymbol{R}$, since they are more computationally intensive during inference and learning than the log-linear factors.

Structured factors We include only one structured factor, PTREE, which constrains the syntactic dependency variables $\boldsymbol{E}$ to form a projective tree. See Section 2.3.3.4 for a detailed description of the form of this factor.

Figure 6.1 depicts the factor graph for a short sentence. The factor graph for our joint model has elements of those given earlier in this thesis: The model of syntactic/semantic dependencies is akin to our models from Section 3.2.3 and Section 5.2. The combination of exponential family factors and the neural network FCM factors is similar to those used in our relation extraction model from Section 4.5.

6.3 Inference

The goal of marginal inference is to compute or approximate the variable and factor marginals (reiterated from equation (2.16) and equation (2.17)):

$$p_{\boldsymbol{\theta}}(y_i \mid \boldsymbol{x}) = \sum_{\boldsymbol{y}' : y'_i = y_i} p_{\boldsymbol{\theta}}(\boldsymbol{y}' \mid \boldsymbol{x}) \tag{6.3}$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}_\alpha \mid \boldsymbol{x}) = \sum_{\boldsymbol{y}' : \boldsymbol{y}'_\alpha = \boldsymbol{y}_\alpha} p_{\boldsymbol{\theta}}(\boldsymbol{y}' \mid \boldsymbol{x}) \tag{6.4}$$

and the partition function (reiterated from equation (2.9)):

$$Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \prod_{\alpha} \psi_\alpha(\boldsymbol{y}_{\boldsymbol{\alpha}}, \boldsymbol{x}) \tag{6.5}$$

Exact inference in our model is intractable due to high treewidth of the factor graph. However, we can carry out approximate marginal inference by structured loopy belief propagation (BP) (Smith and Eisner, 2008). For a detailed discussion of this algorithm, we refer the reader to Section 2.3.3.4, Section 5.2, and Section 5.5.3. Here, we highlight the important characteristics of applying this algorithm to *our* model.

Structured BP is a message passing algorithm, where each message takes the form of a (possibly unnormalized) distribution over a single variable in the factor graph. Messages from structured factors (i.e. those with a large set of neighboring variables) are computed by variants of familiar dynamic programming algorithms—for our model the PTREE factor uses a variant of the inside-outside algorithm of Eisner (1996). The other messages are easily computed with standard tensor operations—these include messages from our log-linear factors and neural factors. These message computations are *local* in that they only look at the incoming messages and, for messages from factors, one potential function.

On acyclic graphs (examples include our SRL and relation extraction models from Chapter 3 and Chapter 4), this algorithm performs exact marginal inference. On cyclic

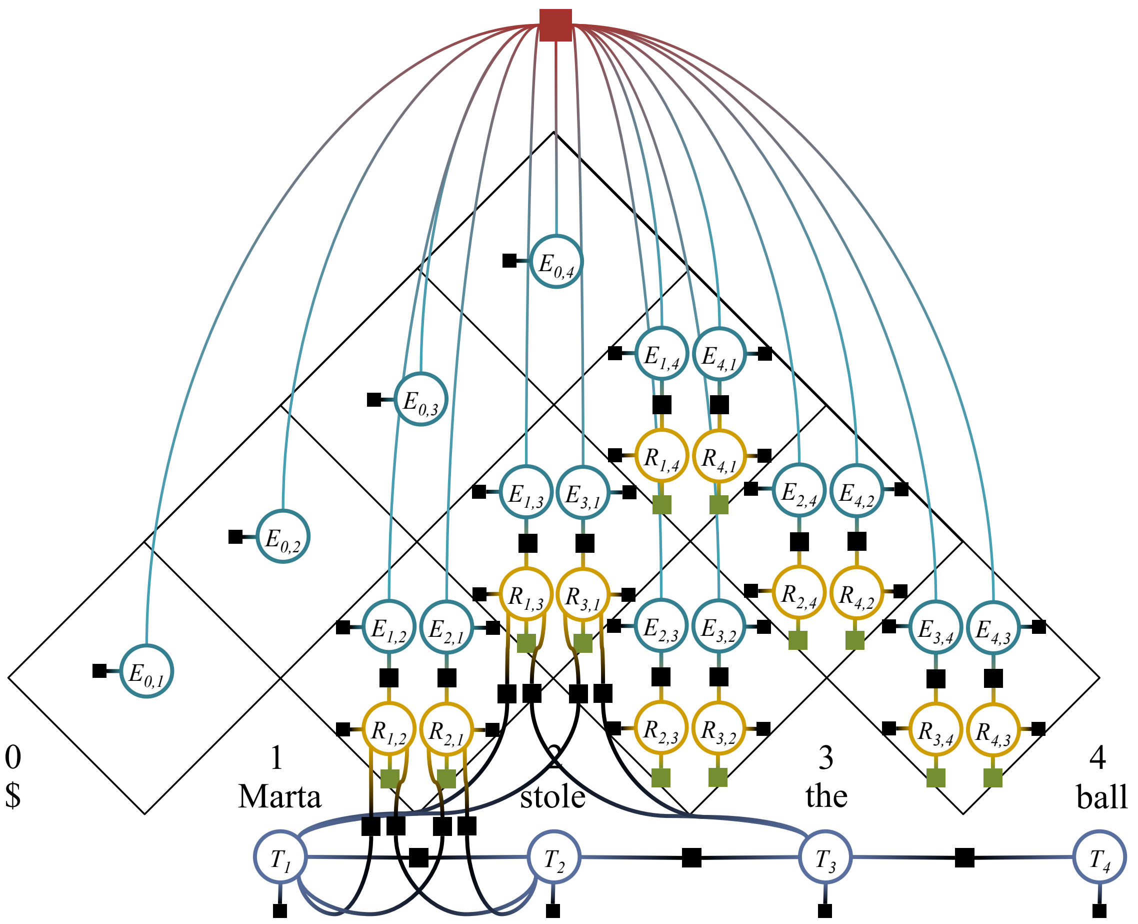


Figure 6.1: Factor graph for joint semantic and syntactic dependency parsing and syntactic tagging of a 4-word sentence; \$ is the root of the dependency graph. The semantic role variable $R_{p,a}$ (yellow) encodes whether and what type of role holds between a predicate p and an argument a. The boolean variable $E_{h,m}$ (blue) encodes whether the syntactic dependency edge from head h to modifier m is present. The tag variable T_i gives the part-of-speech tag for word i. The structured PTREE factor (red) coordinates all the syntactic dependency variables to ensure that the edges form a tree. Each unary FCM factor (green) scores a semantic role variable using a neural network. The remaining factors (black) score one or more variables according to a log-linear function using hand-crafted features. The simplest of these are unary and score each variable in isolation. The binary factors between semantic role and syntactic dependency variables score the syntax/semantics interface. The binary factors between pairs of tag variables score tag bigrams. The drawing shows a few factors between the semantic role variables and the tag variables. Note that the combination of all these factors yields a cyclic ("loopy") graph.

graphs (i.e. those with loops), such as our joint model (Section 6.2), the algorithm performs approximate inference by ignoring the loops. It terminates either at convergence or after a fixed number of iterations. The outputs of BP are beliefs (i.e. approximate variable and factor marginals). The objective functions we consider for training (Section 6.5) will rely on these beliefs.

6.4 Decoding

To facilitate comparison with prior work and to evaluate our models, we wish to obtain a single assignment to the output variables. For the semantic role labeling task we consider in Section 6.6, our true loss function is F1 score: the harmonic mean of precision and recall for the semantic role variables $\boldsymbol{R}$. A minimum Bayes risk (MBR) decoder for this task should take this loss function into account. However, doing so is not straightforward because the loss function doesn't decompose over the factors—by contrast, it is coupled across sentences. For simplicity, we instead use the MBR decoder for Hamming loss (reiterated from equation (2.15)):

$$\hat{r}_i = h_{\boldsymbol{\theta}}(\boldsymbol{x})_i = \operatorname*{argmax}_{\hat{r}_i} \; p_{\boldsymbol{\theta}}(\hat{r}_i \mid \boldsymbol{x}) \tag{6.6}$$

This same decoder was employed in Chapter 3 for SRL and Chapter 4 for relation extraction. In Chapter 5, we also used an MBR decoder for parsing with a constraint that the output variables formed a tree.

6.5 Learning

The training data for SRL in the low-resource setting consist of a dataset of pairs $\{\boldsymbol{x}^{(d)}, \boldsymbol{r}^{(d)}\}_{d=1}^{D}$ where $\boldsymbol{x}^{(d)}$ is a sentence, and $\boldsymbol{r}^{(d)}$ a role labeling. We do not observe either a syntactic dependency parse $\boldsymbol{e}$ or a tagging $\boldsymbol{t}$. The goal of learning is to find model parameters $\boldsymbol{\theta}$ which yield a decision function $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ whose predictions give low loss on the unobserved test sentences. As in Section 5.3, we minimize an ℓ_2-regularized objective function:

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{D}\Big(\big(\sum_{d=1}^{D} J(\boldsymbol{\theta}; \boldsymbol{x}^{(d)}, \boldsymbol{r}^{(d)})\big) + \frac{\lambda}{2}||\boldsymbol{\theta}||_2^2\Big) \tag{6.7}$$

where $\lambda > 0$ is the regularization coefficient and $J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{r}^*)$ is a given differentiable objective function. Our model parameters $\boldsymbol{\theta}$ consist of all those needed for the log-linear and neural factors in our model.

6.5.1 Approximation-Unaware Training

The standard approach to training a graphical model is conditional log-likelihood maximization. We can also apply this technique to our graphical model with structured and neural factors. We set $J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{r}^*) = \log p_{\boldsymbol{\theta}}(\boldsymbol{r} \mid \boldsymbol{x})$ in order to maximize the marginal likelihood in equation (6.1). This log-likelihood is computed as the difference of two partition functions (see Section 2.3.4.1 for details). We can approximate those partition functions using the Bethe Free Energy (see Section 2.3.3.3 for an explanation) which is a simple function of the beliefs output by Structured BP, given in equation (2.27).

Section 2.3.4.1 describes how to compute the gradient of this marginal log-likelihood objective when all of the factors are log-linear. This is not the case in our model, because we include neural factors. Instead, we compute the partial derivatives of the conditional log-likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{r}|\boldsymbol{x})$ with respect to the log potential functions $\log \psi_\alpha(\boldsymbol{y}_\alpha)$. These partials require the true factor marginals, but we replace them with the final beliefs from structured BP. Finally, we backpropagate from these partials through the factors to the model parameters. This gives the gradient of the **surrogate marginal log-likelihood**, the marginal variant of Wainwright (2006)'s surrogate likelihood. This is akin to the surrogate likelihood objective we considered in Section 5.6, yet there we did not *marginalize out* any of the variables.

For some of the models we will consider, structured BP computes the true marginals, in which case we are maximizing the **conditional marginal log-likelihood**.

6.5.2 Approximation-Aware Training

The surrogate likelihood training described above may perform poorly when the inference approximation is poor. Here, we instead consider training in an approximation-aware fashion. Following Section 5.3 we could treat our entire system (inference, decoding, loss) as a differentiable circuit and minimize the regularized empirical risk. We take the simpler approach of minimizing the L_2 distance objective presented in Section 5.4.2 which does not incorporate the decoder or (true) loss function into the system during backpropagation training. That is, we set $J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{r}^*) = \sum_i \sum_{r_i} (b_i(r_i) - b_i^*(r_i))^2$, where the L_2 distance is computed only over the semantic role labeling variables $\boldsymbol{R}$ observed during training.

In Section 6.3, we used a *probabilistic* definition of inference for our graphical model (i.e. two of the three inference tasks from Section 2.3.3). By contrast, inference in a neural network amounts to a straightforward feedforward computation (see examples in Section 2.2) that might have no probabilistic interpretation. By training our approximation-aware model, we have effectively defined a new deep neural network, where inference is a feed-forward computation. Note however, that in *our* deep network, this feed-forward computation incorporates several iterations of BP and any embedded dynamic programming algorithms used to compute messages from structured factors. In this way, inference could be said to have no probabilistic interpretation (we gave this up as soon as we chose to do approximate inference by BP!). However, our inference procedure provides a unified method of combining BP on a factor graph, dynamic programming on a hypergraph, and the feed-forward computation of a neural network. The goal of training is therefore to tune the parameters so that these algorithms perform well in concert with each other.

6.6 Experiments

The goal of our experiments is to explore the merits of graphical models with structured factors, neural factors, and approximation-aware training. To that end, we consider the task of low-resource SRL.

6.6.1 Experimental Setup

Data We consider five languages from the CoNLL-2009 Shared Task (Hajič et al., 2009): Catalan, Czech, German, English, and Spanish. In order to simulate a low-resource setting, for each language, we use only the first 1000 sentences from the training set and discard the rest. We use the standard development and test sets. We also remove all the supervised or automatically annotated data (e.g. lemmas, part-of-speech tags, morphology, dependency trees) except for the words and the semantic roles. Note that our use of the full development set is somewhat artificial for the low-resource setting since its size for most languages is comparable to the training set size. However, using this dev set allows us to carefully regularize our models by early stopping (see below)—thereby improving the stability of our results.

Evaluation Metrics Following the standard evaluation for the shared task, we report Precision, Recall, and F1 on the test set. Each of these can be computed for two settings: *unlabeled* and *labeled*. The unlabeled case assesses whether the correct arguments were identified. The labeled case further asks whether the argument was given the correct role label (arg0, arg1, argM, etc.). Regardless of the evaluation method, we always train on the full labeled training set. These quantities are computed by the standard evaluation script from the shared task.

Hyperparameters The learning rate is selected automatically on a subsample of the training data. The embeddings are rescaled so that $||\boldsymbol{e}||_2 = 1$. The weight of the ℓ_2-regularizer is $\lambda = 1$. We also regularize by early stopping; that is we select the model with the highest labeled F1 on the development set, checking at the end of each epoch.

Models We consider a sequence of models, starting with a baseline and additively building up to our full model.

- **(A)** Our *baseline* SRL model consisting only of the semantic role labeling variables $\boldsymbol{R}$ with unary log-linear factors. This is the SRL-only model from Section 3.2.1.3 and Gormley et al. (2014).
- **(B)** We next add the latent syntactic dependency edge variables $\boldsymbol{E}$ and the binary factors connecting them to the role variables $\boldsymbol{R}$.
- **(C)** This model additionally includes the structured factor, PTREE, which constrains the dependency edge variables $\boldsymbol{E}$ to form a tree. This is the joint SRL model from Section 3.2.3 and Gormley et al. (2014).
- **(D)** Next we add the latent tag variables $\boldsymbol{T}$ and the factors connecting them to the role variables $\boldsymbol{R}$. This is our first *cyclic* ("loopy") *model*. We run BP for only 4 iterations using the same message passing schedule described in footnote 6 of Section 5.2.

(E) We then add the neural factors which score the role variables $\boldsymbol{R}$ according to a log-linear FCM submodel.

(F) Finally, we allow for fine-tuning of the word embeddings, thereby replacing the log-linear FCM submodel with its log-bilinear equivalent.

($\bar{\text{D}}$), ($\bar{\text{E}}$), ($\bar{\text{F}}$) For each loopy model above we also consider the variant trained with approximation-aware learning to maximize the L_2 distance objective function. ($\bar{\text{F}}$) constitutes our *full model*.

Features The feature set we use for the unary and binary log-linear factors on the role $\boldsymbol{R}$ and parse $\boldsymbol{E}$ variables are identical to those described in Section 3.4.2 for the low-resource setting (there they are denoted by IG_C). We do the same feature selection by information gain described in Section 3.2.6.

For the FCM factors we use a feature set that is similar to those given for relation extraction in Table 4.2: In place of the heads of first and second named entity, we consider the predicate and argument heads. In place of the named entity types, we use a brown cluster cutoff to length 4. We consider fewer in-between features: only those up to a maximum of 4 words away from either the predicate or argument heads. Since we do not observe any dependency trees, we do not include the on-path features.

6.6.2 Results

Additive Experiment Our main results, presented in Table 6.1, are an additive experiment on five CoNLL-2009 languages. We compare labeled (Table 6.1b) and unlabeled (Table 6.1a) F1 for 7 models from the sequence of models described in Section 6.6.1. Our aim is to better understand the contributions of different aspects of the full model ($\bar{\text{F}}$). We highlight two baselines from among this sequence: The 'SRL unary only' baseline (A) is the semantics-only model from Section 3.2.1.3. The row '+PTREE factor' (C) corresponds to our joint syntax-semantics model from Section 3.2.3.

Adding the latent syntax tree variables $\boldsymbol{T}$, the PTREE factor, the FCM factor, and approximation-aware training all improve performance. The biggest average gain (+14.82) is given by the addition of the structured factor. Two additions to the model hurt average F1: the addition of the latent tag variables $\boldsymbol{T}$ and the incorporation of fine-tuning. The bulk of the drop in performance when adding the latent tags comes from the German (de) language setting, the annotations for which are very sparse. It seems reasonable that fine tuning would cause the model to overfit—however both the train and dev F1 go down when adding fine tuning. Because the learning rate is automatically selected and we only run for a fixed number of epochs, the lack of overfitting may be evidence that training did not converge. On average, our best model ($\bar{\text{E}}$) (in both labeled and unlabeled F1) is obtained by combining all of the ingredients *except* for fine-tuning.

Precision and Recall on English While our discussion above focused on F1, we also considered the performance of the same sequence of models on precision and recall. Figure 6.2 shows the results for English only. Observe that any increase of more than 0.5 in F1

		Unlabeled F1						
		ca	cs	de	en	es	Avg.	Avg. Diff.
(A)	SRL unary only	42.42	39.26	18.37	46.75	44.12	38.18	–
(B)	+latent tree vars	45.55	45.73	18.79	47.33	47.16	40.91	+2.73
(C)	+PTREE factor	65.51	56.23	31.79	59.01	66.11	55.73	+14.82
(D)	+latent tag vars	66.55	57.09	25.37	59.45	66.56	55.00	-0.73
(E)	+FCM factors	**70.08**	**63.79**	34.63	63.23	70.04	60.35	+5.35
(Ē)	+approx.-aware	70.03	61.95	**39.78**	**63.43**	**72.52**	**61.54**	+1.19
(F̄)	+fine tuning	66.95	57.90	38.13	63.20	69.69	59.17	-2.37

(a)

		Labeled F1						
		ca	cs	de	en	es	Avg.	Avg. Diff.
(A)	SRL unary only	31.99	33.65	13.38	39.56	32.20	30.16	–
(B)	+latent tree vars	33.95	38.26	13.54	39.80	33.83	31.88	+1.72
(C)	+PTREE factor	44.89	43.04	20.95	46.70	44.30	39.98	+8.10
(D)	+latent tag vars	45.42	43.49	18.28	47.51	44.95	39.93	-0.05
(E)	+FCM factors	49.86	**50.90**	24.57	51.36	50.36	45.41	+5.48
(Ē)	+approx.-aware	**50.38**	47.72	**28.37**	**52.94**	**51.86**	**46.25**	+0.84
(F̄)	+fine tuning	47.85	43.65	27.43	50.46	49.40	43.76	-2.50

(b)

Table 6.1: Additive experiment for five languages from CoNLL-2009: Catalan (ca), Czech (cs), German (de), English (en), and Spanish (es). Results on both unlabeled (a) and labeled (b) F1 are shown. We also include the average F1 (Avg.) and the average difference in F1 for each model and the one above it (Avg. Diff.). Details of the models are given in Section 6.6.1.

is always accompanied by an improvement to both precision and recall. The precision is fairly high for all the models and only improves slightly: our baseline (A) obtains precision 80.21 and it increases only to 86.74 with our best model (Ē). By contrast, recall remains low for all of our models, though the increase is larger: the baseline performance of (A) at 32.99 increases to 50.00 for our best model (Ē).

Effects of Approximation-aware Learning Finally, we consider the effects of approximation-aware learning on three different models. The first is our loopy model obtained by including both latent parsing $\boldsymbol{E}$ and tagging $\boldsymbol{T}$ variables and the accompanying factors (D). The second additionally includes the FCM factors on the role variables $\boldsymbol{R}$ (E). The third adds fine-tuning of the word embeddings (F). We contrast surrogate likelihood training (Section 6.5.1) with training by backpropagation with the L_2 distance objective (Section 6.5.2). Training with the latter corresponds to the models (D̄), (Ē), and (F̄) described in Section 6.6.1.

Table 6.2 presents our results on labeled and unlabeled F1. On average, L_2 distance training performs better across the four languages shown than surrogate likelihood training.

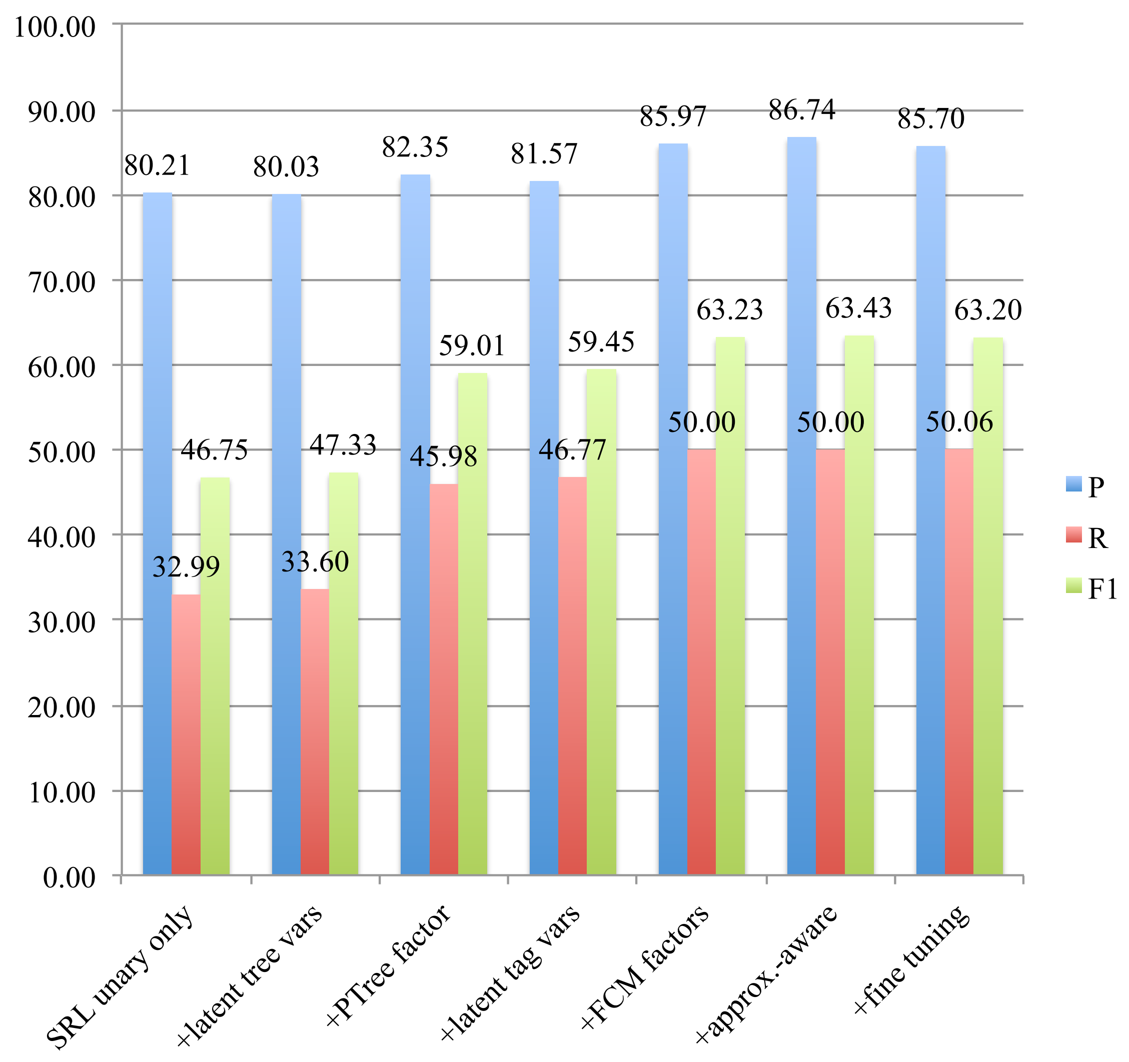


Figure 6.2: Unlabeled precision (P), recall (R), and F1 for additive experiment on English data from CoNLL-2009. The sequence of models and the F1 results are the same as that in Table 6.1a—the P/R results shown here are not given in the table.

		ca		de		en		es		Avg.	
		CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$
(D)	all latent	66.55	-1.77	25.37	+11.22	59.45	-1.89	66.56	-1.09	54.48	+1.62
(E)	+FCM factors	70.08	-0.05	34.63	+5.15	63.23	+0.20	70.04	+2.48	59.50	+1.95
(F)	+fine tuning	68.70	-1.75	23.64	+14.49	61.30	+1.90	68.43	+1.26	55.52	+3.98

(a) Unlabeled F1.

		ca		de		en		es		Avg.	
		CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$	CLL	$L_2 - \text{CLL}$
(D)	all latent	45.42	+0.23	18.28	+5.74	47.51	-1.07	44.95	+0.24	39.93	+0.86
(E)	+ FCM factors	49.86	+0.52	24.57	+3.80	51.36	+1.58	50.36	+1.50	60.35	+1.19
(F)	+fine tuning	47.62	+0.23	17.09	+10.34	49.48	+0.98	48.53	+0.87	40.68	+3.08

(b) Labeled F1.

Table 6.2: Effect of approximation-aware learning. Results are show for both unlabeled (a) and labeled (b) F1. We report *absolute* F1 for the surrogate likelihood baseline (CLL) and the *improvement* in F1 for L_2 over CLL ($L_2 - \text{CLL}$) with positive/negative differences in blue/red.

However, for Catalan (ca), surrogate likelihood *always* performs better in unlabeled F1. Further, for unlabeled F1, most of the gains in that average come from German (de). The gains in labeled F1 are more stable. In all but one case, approximation-aware learning outperforms the baseline.

6.6.3 Error Analysis

In this section, we attempt to isolate the contributions of specific model components on English performance. Specifically, we focus on the two additions to the model that gave the largest gains in F1 on English in our main results: the PTREE factor and the FCM factors. We consider a set of four models:

1. **New Baseline (NB):** This model, trained with approximation-aware learning and an L_2-distance objective, contains the semantic *and* syntactic dependency variables, and their associated unary factors. It also includes binary factors connecting each pair.

2. **NB+PTREE:** This model adds a PTREE factor to the New Baseline. Notice that this new factor does not introduce any additional model parameters since it is a purely declarative constraint over the latent syntactic variables.

3. **NB+FCM:** Next, we take the New Baseline and add the FCM factors. Table 6.3 shows that, unlike PTREE, the FCM factors introduce a very large number of model parameters yielding a much higher capacity model.

4. **NB+PTREE+FCM:** Finally, we combine the PTREE factor and the FCM factors into the New Baseline model.

The experiments in this section mirror the experimental setup described in Section 6.6.1, except that we train on the first 5,000 sentences in the CoNLL-2009 English dataset. The

		Unlabeled			Labeled		
Model	**# Parameters**	**P**	**R**	**F1**	**P**	**R**	**F1**
NB	6,757,191	84.62	51.23	63.82	74.22	44.93	55.98
NB+PTREE	6,757,191	86.37	57.73	69.20	74.85	50.03	59.97
NB+FCM	9,316,039	86.01	57.99	69.28	76.01	51.25	61.22
NB+PTREE+FCM	9,316,039	87.63	62.66	73.07	77.18	55.19	64.36

Table 6.3: Comparison of labeled and unlabeled precision (P), recall (R), and F1 across four models described in Section 6.6.3. Each model is trained on 5,000 sentences from English CoNLL-2009. We also report the number of model parameters for each model considered in the error analysis on English CoNLL-2009. Since the *New Baseline* already includes the latent syntactic variables, adding the PTREE factor (+PTREE) does not increase the number of parameters. By contrast, adding the FCM (+FCM) adds an additional 2.5 million parameters.

development and test sets remain the same. Table 6.3 presents the main results for this setting. First, we observe that Labeled F1 for the best model on these 5,000 training sentences (64.36 F1) is only +0.9 F1 higher than our best model trained on 1,000 sentences (63.43 F1). Next, we turn to a comparison between the two additions to the new baseline: Adding the PTREE factor to New Baseline model (NB+PTREE) yields improvements of +3.99 Labeled F1 and +5.38 Unlabeled F1. Adding the FCM factors (NB+FCM) gives similar improvements: +5.24 Labeled F1 and +5.46 Unlabeled F1. This leads us to contrast the relative benefits of the two very different sorts of factors: a structured PTREE factor with no parameters vs, the high capacity FCM neural factors. While the improvement of NB+FCM over NB+PTREE is noticeable on Labeled F1 (+1.25 F1), it is very minimal on Unlabeled F1 (+0.08 F1). This suggests that incorporating domain-knowledge (e.g. these latent variables should form a projective tree) can be almost as effective as greatly increasing the capacity and generalizability of the model with learned features. Finally, we observe that the two types of factors yield complementary improvements as seen in Section 6.6.2.

Next, we consider three different views of the same results in search of whether the two primary models (NB+PTREE and NB+FCM) under consideration exhibit different patterns of errors.

Predicate-Argument Distance Next, we divide all the possible predicate-argument pairs into bins by the number of tokens separating the predicate head and argument head. For each bin, we compute the F1 score of each model on only the corresponding subset of predicate-argument pairs. These results are summarized in Figure 6.3. The largest *relative* improvements are found on longer dependencies (e.g. 3 to 6 tokens apart, and more than 7 tokens apart) for both NB+PTREE and NB+FCM. However, these relative improvements also correspond to the settings which, without those added factors, were performing the worst. The distribution of the gold predicate-argument pairs between the bins was fairly even: 38.13% separated by 1 token, 23.74% by 2 tokens, 25.05% by 3-6, and 13.08% by 7 or more.

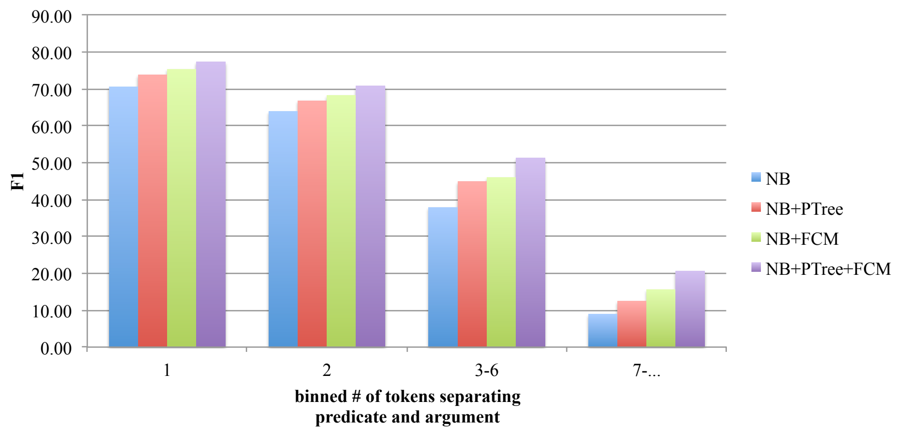


Figure 6.3: F1 of SRL for predicate-argument distance. We divide each possible predicate-argument pair into a bin based on the number of tokens separating the two heads: 1 token, 2 tokens, 3 to 6 tokens apart, or 7 or more tokens. The F1 computation is restricted to only the semantic edges in the respective bin for the four models in Section 6.6.3.

Nominal vs. Verbal Predicates We can also divide the predicate-argument pairs by the (gold) part-of-speech tag for the predicate head. The full set of such Penn Treebank tags when truncated to the first two characters includes CC, CD, IN, JJ, NN, PD, RP, VB, WP, and WR. However, 99.70% of them are accounted for by the nominal (NN, 39.47%) and verbal (VB, 60.23%) predicates. So we focus our discussion only on these two types of predicates. Figure 6.4 gives the F1 results for our four models binning by whether the predicate was nominal or verbal.

Despite there being over 1.5 times as many verbal predicate-argument training examples, each model performs respectively better on nominal predicates than verbal. We find that relative improvement of NB+FCM over NB+PTREE is much higher on the nominal than verbal predicates.

Semantic Role Types Finally, we ask whether there are observable differences in the relative improvements across role labels for the two types of factors. We again bin the pairs, this time by the label of the predicate-argument pair. These post-hoc results are akin to what we would observe if we trained a separate model for each role. One of the smallest differences in the relative improvement given by +PTREE and +FCM is found in the most frequently occurring role, A1, which usually corresponds to an Patient or Theme role. The advantage of +FCM over +PTree seems particularly pronounced by the second most common role, A0, which is often an Agent role.

6.7 Summary

In this chapter, we present graphical models with structured factors, neural factors, and approximation-aware training. We introduce a model for joint semantic role labeling, syntactic dependency parsing, and part-of-speech tagging. By treating the syntax as latent, we

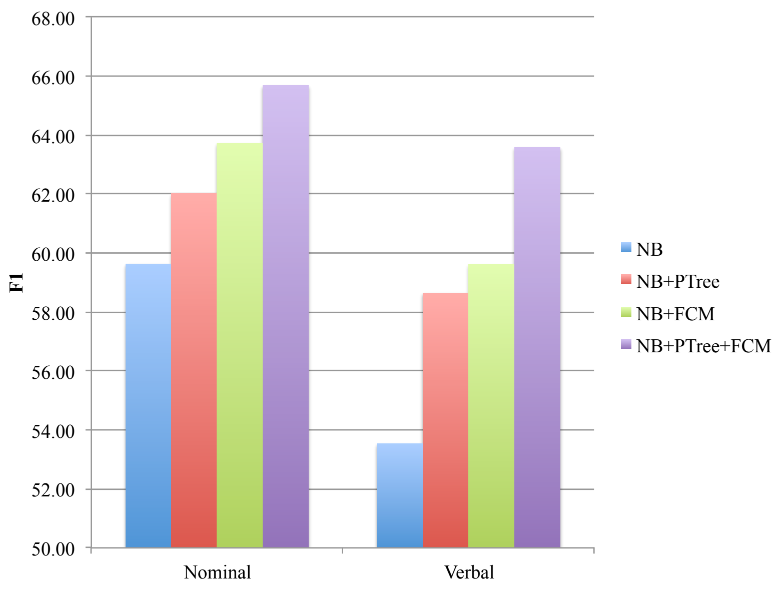


Figure 6.4: F1 of SRL across nominal and verbal predicates. We bin the predicate-argument pairs based on whether the predicate is nominal (has a gold POS tag starting with NN) or verbal (POS tag starting with VB). F1 is computed separately for each bin on each of the four models in Section 6.6.3.

can train in a semi-supervised fashion where only the semantics are observed. We find that structured factors, neural factors, and our training method all improve performance over our baseline models.

Role Label	% of Gold	NB	NB+PTREE	NB+FCM	NB+PTREE+FCM
A1	37.06%	59.21	64.05	64.75	68.44
A0	25.15%	60.41	63.35	64.90	67.81
AM	20.78%	45.75	50.93	52.46	55.99
A2	11.31%	54.67	56.44	57.69	59.89
A3	2.22%	51.84	51.83	54.02	54.55
R-	2.07%	48.13	54.55	61.42	61.99
C-	0.88%	48.23	56.64	54.60	57.14
A4	0.50%	56.67	57.59	60.00	62.00
A5	0.03%	0.00	0.00	0.00	0.00

Table 6.4: F1 of four models from Section 6.6.3 across role labels. For each row, we treat all but one label (Role Label) as corresponding to the nil label. We take only the first two characters of the role so that the many various roles starting with AM- are combined under the row AM. We report the results ordered by the proportion of each role appearing in the gold data (% of Gold).

Chapter 7

Conclusions

7.1 Summary of the Thesis

The primary contribution of this thesis was the introduction of graphical models with structured factors, neural factors, and approximation-aware training.

Chapter 3 We presented the most thorough study to date of semantic role labeling in low-resource settings. We introduced distant semantic supervision for grammar induction by way of a constrained E-step in Viterbi EM. Further, we presented the first empirical study of joint vs. pipelined training of SRL with latent syntax. Our alteration of the model from Naradowsky et al. (2012a) obtained the best results, and strong results in the fully supervised setting. Our results indicate that for SRL in the low-resource setting joint models can outperform pipelined models.

Chapter 4 We investigated the role of neural and handcrafted features on relation extraction. Our primary finding was that the two types of features are highly complementary in relation extraction when using the FCM of Gormley et al. (2015b). We obtained state-of-the-art results on ACE 2005 relation extraction in a domain adaptation setting. Our results on SemEval-2010 Task 8 relation classification approach the best reported result on that benchmark.

Chapter 5 We introduce approximation-aware learning for structured belief propagation (BP)—an extension of the ERMA method of Stoyanov et al. (2011) to structured factors. We further introduce a new objective function based on the L_2 distance between the beliefs and the one-hot representation we *want* them to take on. Our results demonstrate that our method trains parsers that are faster and more accurate than those trained by traditional conditional log-likelihood.

Chapter 6 We present a new framework for hybrids of graphical models with structured factors and neural networks. When the factor graph contains cycles, our method treats the forward pass through a neural network, approximate inference, any embedded dynamic programming algorithms, decoding, and loss as defining a deep network, which can be trained by backpropagation. We apply this method to a new model for joint syntactic and semantic dependency parsing.

7.2 Future Work

This section mentions a few of the possible directions for extending and building upon this work.

7.2.1 Other Structured Factors and Applications

In Chapter 5 we only considered second order dependency parsing, however the third-order features of Martins et al. (2013) could be adapted to our framework. Further, we could consider neural features akin to recent work in neural networks for dependency parsing (Chen and Manning, 2014).

While this work has focused on *dependency* structures, there are many other applications that we could consider. For example, most existing applications of structured BP would likely benefit from our approach. Structured BP has already been applied to CNF grammar parsing (Naradowsky et al., 2012b), TAG parsing (Auli and Lopez, 2011), an ITG-constraint for phrase extraction (Burkett and Klein, 2012), graphical models over strings (Dreyer and Eisner, 2009), and taxonomy induction (Bansal et al., 2014)—among others.

Other areas for application include computer vision tasks such as scene parsing, pose estimation, and image captioning. In computational biology, the problems of folding, aligning, and modeling RNA sequences also provide a natural problem space for the types of models proposed here.

7.2.2 Pruning-aware Learning

Multi-pass coarse-to-fine inference has proven to be a very effective method for tasks in NLP such as constituency parsing (Petrov et al., 2006; Petrov and Klein, 2007; Pauls and Klein, 2009) and machine translation (Petrov et al., 2008; Petrov, 2009). Traditionally, these approaches have relied on maximum likelihood training of the coarse models. Structured prediction cascades (Weiss and Taskar, 2010) instead define an objective function for each intermediate pruning model that encourages a high oracle pruning accuracy. Applied to MAP inference for dependency parsing (Rush and Petrov, 2012) these structured prediction cascades lead to significant speedups with minimal loss in accuracy.

A natural extension of our work is to treat the complete sequence of pruning models and the final decoder as a single (approximate) system. By carefully defining the pruning decisions by a *sub*differentiable step function, we could backpropagate through them just as we would any other part of our model. The pruning would be active not just at test time, but also during training—so that both would see efficiency gains.

Newswire:	President elect Mohammed Morsi leads the ”Freedom Justice Party” (FJP), an emanation of the Muslim Brotherhood
Twitter:	b/c egypt’s morsi chaired the fjp!!!

Table 7.1: Example sentences from newswire and Twitter domains.

7.2.3 Hyperparameters: Optimizing or Avoiding

A deficiency of the methods in this thesis—as with deep learning—is the need for tuning of hyperparameters. In this work, we relied on manual tuning, grid search, and random search (Bergstra et al., 2011; Bergstra and Bengio, 2012) for hyperparameter optimization. Yet more sophisticated methods, such as tree-structured Parzen estimators (Bergstra et al., 2011) or Gaussian process optimization (Snoek et al., 2012) would likely yield better results. A particularly complementary approach would be the efficient backpropagation method of Maclaurin et al. (2015), which treats hyperparameters as another tunable weight in the system.

Hyperparameter optimization should not be left to guesswork. It should be treated as an essential part of the scientific process (Bergstra et al., 2013). Our strong emphasis on continuous optimization in Section 2.4 was (in part) because *choosing* the right optimization algorithm was an important part of our process of hyperparameter optimization. Thus, we take the position that careful work in this area is just as important as any of the other extensions mentioned here.

Since many of the most important hyperparameters relate to the learning algorithm, an alternative would be to consider the algorithms that have fewer (or at least less sensitive) hyperparameters. For example, the online learning method of Martins et al. (2010b) and Martins et al. (2010a) could possibly be adapted to the approximation-aware setting.

7.2.4 Multi-task Learning for Domain Adaptation

Most natural language processing (NLP) tools are brittle: having been trained on one language, style, and domain, the quality of their annotations erodes when transferred to a different setting, and ad-hoc domain adaptation techniques help only slightly. Consider transferring across writing styles from newswire to Twitter data (see Table 7.1). We would expect that the most prominent changes will come about in spelling, where letters and sometimes entire words are dropped. To a lesser extent we anticipate the syntax to change. If the text emphasizes sports, we might expect that the entities and relations discussed will change relatively little. The stability of these things is what allows us to puzzle out the most plausible interpretation, despite many changes to orthography and relatively few to syntax and the facts. In this sense, the correct interpretation would be *overdetermined* in the correct model.

One of the primary motivations for this work was the goal of jointly modeling multiple linguistic strata: orthography, syntax, shallow semantics, topic, and knowledge. Model parameters can then be tied across styles/genres in a hierarchical Bayesian setting. This would allow the model to transfer only the appropriate levels of linguistic knowledge, while learning which parameters must adapt to account for variation across these settings. Critically, our model will allow for confidence in one level to propagate to all the others. For example, we might not know how spelling works in one setting, so we rely on a higher level of the model to figure it out. The learned constraints on language are propagated across two different axes inherent in the data: linguistic strata (e.g. semantics to orthography) and domains (e.g. newswire to twitter). The value proposition is that if our model knows about more constraints on language, it can better withstand and adapt to perturbations of the data.

Learning in this model would likely take on a semi-supervised form. Out-of-domain annotated data will be essential to guide learning in its early stages. Yet we will decrease its influence as we gradually build confidence on the in-domain data, marginalizing over the levels of the model for which there is the most uncertainty. Consider a case where our target domain is weblogs, for which we have only relation annotations. Parameter estimation would also utilize data from other domains such as a newswire treebank and named entity annotations on broadcast news; allowing the parameters for these domains to influence the marginalized parses and named entities on the weblogs.

Appendix A

Pacaya: A General Toolkit for Graphical Models, Hypergraphs, and Neural Networks

Graphical models, neural networks, and inference on hypergraphs are traditionally treated as distinct. This is reflected in the numerous software frameworks that handle one of the three in isolation. See the related work section below for examples. By contrast, **Pacaya**[1] is a framework for hybrids of graphical models and neural networks, which perform approximate inference with hypergraph algorithms as a subroutine. The design and engineering of this framework was critical to the study of the new types of models discussed in this thesis.

A.1 Code Layout

The Pacaya framework is made up of four Java libraries:

Prim Prim is a Java primitives library with an emphasis on sparse representations of vectors and matrices. Unlike C++, the Java compiler does not provide built-in support for templates/generics over primitives data types (e.g. `int`, `long`, `float`, `double`). Prim instead uses a canonical definition of a class as a template for code generation. For example, a sorted map is parameterized by the key type (K) and the value type (V). The canonical class is the one where (K, V) = (`long`, `double`). Code generation uses this canonical version to create the sorted map classes for other pairs: (`int`, `float`), (`short`, `int`), etc.

Optimize This library provides a variety of modern algorithms for numerical optimization. The primary focus of the library is on variants of SGD, which have proven to be critical to training large-scale machine learning systems. It includes several of the algorithms described in Section 2.4. The Optimize library easily allows one to switch between optimizers at training time. (A separate library provides wrappers to optimization routines from other toolkits e.g. quasi-Newton methods.)

[1]https://github.com/mgormley/pacaya

Pacaya This library is the core of the framework. Pacaya is a Java library for hybrid graphical models and neural networks. Just like other *neural net* libraries, Pacaya implements module-based automatic differentiation (AD). The novelty in Pacaya is that it includes modules which are a departure from the usual building blocks of neural networks: such as modules for approximate inference by BP, inside-outside on a hypergraph, MBR decoding – these tend to be very sparse. It also includes some more standard NN modules that manipulate dense tensors. Unlike most other *graphical models* libraries, Pacaya was designed to support arbitrary factors (structured, neural). Such factors act as just another module (in the autodiff sense). In this thesis, we consider models where a neural network feeds into approximate inference which calls out to exact inference on a hypergraph. However, the framework would permit other architectures as well, such as approximate inference feeding forward into a neural network.

Pacaya NLP Applications of Pacaya to natural language processing (NLP) reside in this library. The part-of-speech tagger (Chapter 6), dependency parser (Chapter 5), semantic role labeler (Chapter 3), and relation extractor (Chapter 4) are all included. They can be trained and tested as individual components or as a single joint model.

A.2 Feature Sets from Prior Work

Many of the models we consider in this thesis are either identical to or inspired by prior work. Pacaya NLP includes a number of feature sets from these models.

SRL For SRL, we include the features from Björkelund et al. (2009), Zhao et al. (2009), and Naradowsky et al. (2012a). We also include most of the features from Johansson (2009) and Lluís et al. (2013), with missing features noted in the code.

Dependency Parsing We re-implement the syntactic dependency parsing feature sets of McDonald et al. (2005), McDonald and Pereira (2006), Carreras (2007), and Koo et al. (2008). We also include the first- and second- order features from Martins et al. (2013). The library does not (yet) support consecutive sibling factors.

Relation Extraction For relation extraction, we re-implement the features from Zhou et al. (2005) and Sun et al. (2011) with the exception of the relation-specific features requiring a country list, trigger word list, and title list.

Pacaya NLP includes a feature template language that is catered to extracting these sorts of features. In discriminative models, it is common to distinguish between *features* and *properties*. As noted in Sutton and McCallum (2007), features can be defined using a function of the form: $f_{\alpha,\tilde{\boldsymbol{y}}_\alpha,k}(\boldsymbol{y}_\alpha, \boldsymbol{x}) = \mathbb{I}(\tilde{\boldsymbol{y}}_\alpha = \boldsymbol{y}_\alpha) g_k(\boldsymbol{x})$, where $\mathbb{I}$ is the indicator function, $\tilde{\boldsymbol{y}}_\alpha$ is a fixed assignment to the variables, and g_k extracts the kth property of the observations.[2] The vector of features becomes:

$$
\begin{aligned}
\boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}) = [&f_{\tilde{\boldsymbol{y}}_1,1}(\boldsymbol{y},\boldsymbol{x}), f_{\tilde{\boldsymbol{y}}_1,2}(\boldsymbol{y},\boldsymbol{x}), f_{\tilde{\boldsymbol{y}}_1,3}(\boldsymbol{y},\boldsymbol{x}), \dots, &\text{(A.1)}\\
&f_{\tilde{\boldsymbol{y}}_2,1}(\boldsymbol{y},\boldsymbol{x}), f_{\tilde{\boldsymbol{y}}_2,2}(\boldsymbol{y},\boldsymbol{x}), f_{\tilde{\boldsymbol{y}}_2,3}(\boldsymbol{y},\boldsymbol{x}), \dots, &\text{(A.2)}\\
&f_{\tilde{\boldsymbol{y}}_3,1}(\boldsymbol{y},\boldsymbol{x}), f_{\tilde{\boldsymbol{y}}_3,2}(\boldsymbol{y},\boldsymbol{x}), f_{\tilde{\boldsymbol{y}}_2,3}(\boldsymbol{y},\boldsymbol{x}), \dots] &\text{(A.3)}
\end{aligned}
$$

where the α subscripts have been dropped for readability, and $\tilde{\boldsymbol{y}}_i$ is the ith configuration of the variables in factor α. Pacaya NLP provides a little language for defining the property extractors $g_k(\boldsymbol{x})$.

The documentation for Pacaya NLP describes where to find feature sets from prior work in the code. They are implemented declaratively in the little language, imperatively when speed is particularly important (e.g. dependency parsing), and in some cases both declaratively and imperatively.

A.3 Design

A.3.1 Differences from Existing Libraries

There are a variety of other excellent libraries for graphical models, neural networks, and hypergraphs. These include but are by no means limited to the following:

- Graphical model libraries:
  - Factorie (Scala) (McCallum et al., 2009)
  - LibDAI (C++) (Mooij, 2010)
  - OpenGM (C++) (Andres et al., 2012)
  - Infer.NET (.NET) (Minka et al., 2012)
- Neural network libraries:
  - Torch7 (Lua) (Collobert et al., 2011a)
  - Theano (Python) (Bergstra et al., 2010; Bastien et al., 2012)
  - Caffe (C++) (Jia et al., 2014)
- Hypergraph libraries:
  - Pydecode[3] (Python)
  - cdec[4] (C++) (Dyer et al., 2010)

[2] Sutton and McCallum (2007) refer to g_k as an *observation function*.
[3] `https://github.com/srush/PyDecode`
[4] hypergraphs for machine translation decoding

Many of these libraries represent the state-of-the-art for machine learning technology. They are also built on certain restrictive assumptions that made them unsuitable for the goals of this work. For example, the graphical models libraries are designed to support factors of only a few variables, while we needed to support structured factors of many variables. The neural network libraries are built to do very fast processing of dense Tensors, yet they don't readily support the sorts of sparse data structures needed in order to treat inference as a feed-forward network. The hypergraph libraries are likely suitable for our needs, yet it only represents a small portion of the overall codebase. Accordingly, we designed Pacaya from the ground up with the overall design goal of hybrid models in mind. Of course, Pacaya has its own restrictive assumptions, and we discuss some of these below.

A.3.2 Numerical Stability and Efficient Semirings in Java

Numerical stability is an important consideration for both the forward computation and backpropagation of approximate inference in a factor graph. Following Li and Eisner (2009), we rely on a semiring that represents each real number as a pair containing the log of the absolute value of the number and a sign bit. This representation permits very small positive *and* negative numbers. We extend this semiring to its equivalent abstract algebra, in order to accomodate the broader scope of elementary operations we need (add, subtract, times, divide, exp, log, etc.).

Since these operations are often at the most deeply nested inner loops it is important that they be inlined and compiled by the Java Virtual Machine's (JVM) just-in-time (JIT) compiler. We implement each abstract algebra as an object with methods for each of the elementary operations. The data itself is always stored in a double. With careful use of bit shifts and masking, we can carve up the 64-bits into (very primitive) data structures such as two floats, or a log-absolute value and a sign bit. Unpacking, processing, and repacking the bits in this way can be inlined in most modern JVM's whenever exactly one class implementing the abstract algebra interface is loaded – since the JVM can rule out any other possible code paths. However, we often reuse the same data structure (e.g. a tensor object) with two abstract algebras (e.g. *log-sign* and *real*). Thus, there may be two possible branches that could be taken. Modern JVM's support bimorphic inlining, which handles exactly this case efficiently. Unfortunately, current JVM's do not support megamorphic inlining (i.e. inlining three or more possibilities) – so we generally avoid that setting.

A.3.3 Comments on Engineering the System

Maximizing speed, minimizing memory usage, and handling a variety of architectures (CPU and GPU) are some of the main considerations that influence the early design choices of any framework for graphical models, neural networks, or hypergraphs. The frameworks mentioned above prioritize speed of certain dense matrix computations that are particularly useful for deep feed-forward neural networks. Pacaya prioritizes speed of the sparse computations for inference by belief propagation on a factor graph and inside-outside on a hypergraph.

However, the choice of Java over other languages is a concession in speed/memory in favor of portability, quality of tooling (IDEs, debuggers, profilers, etc.), and the flow of an interpreted scripting language (with the Eclipse compiler). While speed comparisons between languages are very nuanced, we present two here that give a flavor for the speed tradeoffs that exist between Java and C++.

A.3.3.1 Experiment 1: Inside-Outside Algorithm

The purpose of this experiment was to assess the differences in speed available to the C++ and Java programmer implementing the inside algorithm for dependency parsing (Eisner, 1996). A key aspect of any parser implementation is the representation of the parse chart. For a dependency parser this can be viewed as a four dimensional table (parent, child, direction, complete) of size $n \times n \times 2 \times 2$. We consider the effect that this has on parser speed.

Setup In Java, all arrays *must* be heap allocated—stack allocation only exists for primitives. However, the size of any array (even high-dimensional arrays) can be specified at runtime. A Java **4D array** permits multidimensional indexing. In C++, a **4D vector** of the type `vector<vector<vector<vector<double>>>>` permits multidimensional indexing and can be sized at runtime. In both Java and C++, we can also use a **1D array** allocated on the heap with the indexing computed by 3 multiplications and 3 additions.[5] Java was compiled with Eclipse Luna's JDT and one round of warm-up was given to the JIT compiler. C++ was compiled with clang v6.0 and either included debug symbols (☑) or did not (☐). The latter case used `-O3` compiler optimizations.

Results Table A.1a compares the speed of these Java and C++ parsers with a max/+ semiring and backpointers to recover the Viterbi parse. We report the average number of tokens per second on 10,000 runs of the inside algorithm using synthetic sentences of length 30. Not surprisingly, the 1D array implementations give a significant speedup over their 4D counterparts. C++ is only somewhat faster than the Java Viterbi parser.

Table A.1b repeats the same experiment running the inside-outside algorithm with a +/log-add semiring and no backpointers. The implementation in log-add for both languages relies on a native call to `log1p`.[6] This operation is dramatically faster in C++ than Java and dramatically effects the results. The parse chart representation is no longer as important as the language choice in this case.

These results suggest that for these algorithms, if test-time performance is the end-goal, then C++ exhibits a clear advantage over Java. However, if a balance between test-time and debug performance is desired, Java should be preferred.

[5]We also tested a true multidimensional array in C++, which must have its dimensions specified at compile time. The advantage of this method is that the parse chart can be allocated on the stack. However, this comes with a disadvantage that the parser cannot parse sentences beyond a fixed length—so we do not include those results here. The speedup over the 1D array was about a factor of 2 for the max/+ semiring and gave no observable speedup for the +/log-add semiring.

[6]In Pacaya, we often approximate `log1p` with a lookup table for additional speed, though we don't report those results here since it degrades numerical stability.

Language	Debug	Storage	Tok./Sec.
C++	☑	4D vector	9,576
C++	☐	4D vector	20,648
C++	☑	1D array	23,997
Java	☑	4D array	54,044
Java	☑	1D array	186,567
C++	☐	1D array	270,459

(a) Viterbi parser with max/+ semiring and backpointers

Language	Debug	Storage	Tok./Sec.
C++	☑	4D vector	1,853
Java	☑	4D array	3,017
Java	☑	1D array	3,401
C++	☐	4D vector	3,639
C++	☑	1D array	5,039
C++	☐	1D array	9,710

(b) Inside-outside algorithm with +/log-add semiring

Table A.1: Speed comparison of Java and C++ parser implementations. The tokens per second were averaged over 10,000 trials for max/+, and over 1,000 trials for +/log-add.

Framework	Language	Algorithm	Total Seconds
LibDAI	C++	BP (`DAI_BP_FAST`=0)	25.25
Pacaya	Java	BP (standard)	19.68
Pacaya	Java	BP (divide-out)	11.45
LibDAI	C++	BP (`DAI_BP_FAST`=1)	10.38

Table A.2: Speed comparison of Pacaya (Java) and LibDAI (C++) implementations of belief propagation (BP) with parallel message passing.

A.3.3.2 Experiment 2: Parallel Belief Propagation

In this section, we compare two implementations of belief propagation with parallel message passing: Pacaya (Java) and LibDAI (C++) (Mooij, 2010). The two libraries exhibit significant differences in the implementation of the algorithm and the choice of data structures. However, Pacaya took inspiration from LibDAI in various design choices. One notable difference is that LibDAI is optimized for pairwise MRFs and does not cache the messages from variables to factors.

Setup LibDAI implements the inner-loops of the message passing algorithm in two ways: one uses a tensor data structure and is very readable (**`DAI_BP_FAST=0`**) and the other is highly optimized for speed (**`DAI_BP_FAST=1`**). Pacaya can optionally cache variable and factor beliefs during message passing which allows messages to be computed by dividing out a message from the beliefs. We consider two versions of Pacaya: one with the dividing out trick (**divide out**) and one without (**standard**). Each framework performs 10 iterations of parallel BP. The same compilers were used as in the above experiment. In order to warm-up the JVM, both implementations were run for 3 trials and the time on the third trial was used. We test on a single factor graph. The model is from Chapter 5 and corresponds to a 2nd-order parser with unary, grandparent, and arbitrary sibling factors. We do not include the PTREE factor since LibDAI does not support structured factors. No pruning was used, so each edge variable is connected to $O(n)$ other edges by grandparent and sibling factors.

Results The results are summarized in Table A.2. The standard implementation in Pacaya is slightly faster than the "readable" implementation in LibDAI, but 2x slower than the optimized version. Using the divide-out trick on this particular factor graph gives a significant speedup, such that Pacaya is almost as fast as LibDAI.

Appendix B

Bethe Likelihood

In this section, we propose an objective function which has the log-likelihood as a special case. We refer to this objective as the **Bethe log-likelihood** because it is identical to the log-likelihood except that we replace the true partition function Z with its Bethe Free Energy approximation Z_{Bethe}.

$$\log p(\boldsymbol{y}) = \sum_{\alpha} \log \psi_\alpha(\boldsymbol{y}_\alpha) - \log Z_{\text{Bethe}} \tag{B.1}$$

We define $-\log Z_{\text{Bethe}} = F_{\text{Bethe}}(\boldsymbol{b})$ where $F_{\text{Bethe}}(\boldsymbol{b})$ is the Bethe Free Energy. When inference is exact, the Bethe Free Energy is equal to the negative log partition function: $F_{\text{Bethe}}(\boldsymbol{b}) = -\log Z$, and in this case the Bethe log-likelihood recovers log-likelihood.

Backpropagating through the first term of the Bethe likelihood is simple. We add to the adjoints of the potential function for each $\boldsymbol{y}_\alpha$ that we observe in the training data:

$$\eth\psi_\alpha(\boldsymbol{y}_\alpha) \mathrel{+}= \frac{\eth p(\boldsymbol{y})}{\psi_\alpha(\boldsymbol{y}_\alpha))} \tag{B.2}$$

Next we consider how to do the forward and backward computations for the Bethe free energy.

Bethe Free Energy with Structured Factors The Bethe Free Energy is computed as a function of the beliefs:

$$\begin{aligned} F_{\text{Bethe}}(\boldsymbol{b}) = & \sum_{\alpha} \sum_{\boldsymbol{y}_\alpha} b_\alpha(\boldsymbol{y}_\alpha) \log \left[\frac{b_\alpha(\boldsymbol{y}_\alpha)}{\psi_\alpha(\boldsymbol{y}_\alpha)} \right] \\ & - \sum_{i} (N_i - 1) \sum_{y_i} b_i(y_i) \log b_i(y_i) \end{aligned} \tag{B.3}$$

For most factors α, this computation is straightforward. However, for the PTREE factor, we require a more efficient method of computing the summation over assignments $\boldsymbol{y}_\alpha$. This reduces to the expected log-beliefs (i.e. entropy) of the distribution over trees for that factor. Li and Eisner (2009) show that this can be computed as a simple linear function of the tree marginals.

Backpropagation through the Bethe Free Energy (B.3) is very simple for the variable beliefs and those of the factors for which we can compute the expected log belief by brute force.

$$\eth b_i(y_i) \mathrel{+}= \eth F_{\text{Bethe}}(\boldsymbol{b})(\log b_i(y_i) + 1) \tag{B.4}$$
$$\eth b_\alpha(\boldsymbol{y}_\alpha) \mathrel{+}= \eth F_{\text{Bethe}}(\boldsymbol{b})(\log b_\alpha(\boldsymbol{y}_\alpha) + 1 - \log \psi_\alpha(\boldsymbol{y}_\alpha)) \tag{B.5}$$
$$\eth \psi_\alpha(\boldsymbol{y}_\alpha) \mathrel{+}= \eth F_{\text{Bethe}}(\boldsymbol{b}) \frac{b_\alpha(\boldsymbol{y}_\alpha)}{\psi_\alpha(\boldsymbol{y}_\alpha)} \tag{B.6}$$

However, we cannot simply enumerate the belief table for structured factors like PTREE. Next we consider how to deal with this issue.

Expected Log Beliefs for PTREE To compute the term $\sum_{\boldsymbol{y}_\alpha} b_\alpha(\boldsymbol{y}_\alpha) \log \left[\frac{b_\alpha(\boldsymbol{y}_\alpha)}{\psi_\alpha(\boldsymbol{y}_\alpha)}\right]$ for $\alpha = \text{PTREE}$, we first observe that we can drop the value $\psi_\alpha(\boldsymbol{y}_\alpha)$ since it always has value 1.0 except when $b_\alpha(\boldsymbol{y}_\alpha)$ is also zero. We then find that these expected log beliefs are just the negative entropy of the distribution over derivations for the hypergraph given by b_{PTREE}:

$$-H(b_\alpha) = \sum_{\boldsymbol{y}_\alpha : \psi_\alpha(\boldsymbol{y}_\alpha)=1} b_\alpha(\boldsymbol{y}_\alpha) \log b_\alpha(\boldsymbol{y}_\alpha) \tag{B.7}$$

where $\alpha = \text{PTREE}$. Computing this negative entropy term requires running the inside-outside algorithm, followed by a simple iteration over the hyperedges (Li and Eisner, 2009). To see that this is the case, we can rewrite the negative entropy as below:

$$-H(b_\alpha) = \sum_{\boldsymbol{y}_\alpha : \psi_\alpha(\boldsymbol{y}_\alpha)=1} \frac{q_\alpha(\boldsymbol{y}_\alpha)}{Z_q} \log \frac{q_\alpha(\boldsymbol{y}_\alpha)}{Z_q} \tag{B.8}$$
$$= \frac{\bar{q}}{Z_q} - \log Z_q \tag{B.9}$$

where $q_\alpha(\boldsymbol{y}_\alpha) = \prod_{i:y_i=\text{ON}} \frac{m_{i\to\alpha}^{(t\max)}(\text{ON})}{m_{i\to\alpha}^{(t\max)}(\text{OFF})}$ are the message ratios, $Z_q = \sum_{\boldsymbol{y}_\alpha : \psi_\alpha(\boldsymbol{y}_\alpha)=1} q_\alpha(\boldsymbol{y}_\alpha)$ is the partition function computed by the inside-outside algorithm on a hypergraph, and $\bar{q} = \sum_{\boldsymbol{y}_\alpha} q_\alpha(\boldsymbol{y}_\alpha) \log q_\alpha(\boldsymbol{y}_\alpha)$. Notice that we have played slight of hand with the product of all the OFF messages, which is implicitly included in Z_q. Following Li and Eisner (2009) we compute $\bar{q}$ by running the inside-outside algorithm with hyperedge weights w_e. Here we use the same hypergraph structure used to compute the beliefs for PTREE and the same hyperedge weights. Namely, a hyperedge e which corresponds to $y_i = \text{ON}$ has weight $w_e = \frac{m_{i\to\alpha}^{(t\max)}(\text{ON})}{m_{i\to\alpha}^{(t\max)}(\text{OFF})}$. Any other hyperedge e has weight $w_e = 1$. Finally, we compute the following from the the inside and outside scores, β_e, α_e, and the logs of the hyperedge weights:

$$\bar{q} = \sum_e \alpha_{H(e)} \log w_e \prod_{j \in T(e)} \beta_j \tag{B.10}$$

to obtain the desired quantity.[1]

[1]The alternative approach would be to run the inside algorithm with a first-order expectation semiring where the hyperedge weights are $\langle w_e, w_e \log w_e \rangle$ (Li and Eisner, 2009).

In order to backpropagate through the expected log beliefs, we assume access to the adjoint of the negative entropy $\eth(\text{-}H(b))$. The computation then proceeds as below:

$$\eth\bar{q} \mathrel{+}= \eth(\text{-}H(b))\frac{1}{Z_q} \tag{B.11}$$

$$\eth Z_q \mathrel{+}= \eth(\text{-}H(b))\frac{\bar{q}}{Z_q^2} - \frac{1}{Z_q} \tag{B.12}$$

Since we can already backpropagate through the inside-outside algorithm, we only need to define the contribution to the adjoints made by the simple computation of $\bar{q}$ in (B.10). The values below act as the *initial* values of the adjoints when running the backward pass through inside-outside.

$$\eth\alpha_i \mathrel{+}= \sum_{e\in I(i)} \eth\bar{q} \log w_e \prod_{j\in T(e)} \beta_j \tag{B.13}$$

$$\eth\beta_j \mathrel{+}= \sum_{e\in O(j)} \eth\bar{q}\, \alpha_{H(e)} \log w_e \prod_{k\in T(e):k\neq j} \beta_k \tag{B.14}$$

$$\eth w_e \mathrel{+}= \eth\bar{q}\, \alpha_{H(e)} \frac{1}{w_e} \prod_{j\in T(e)} \beta_j \tag{B.15}$$

Recall that the input to this inside-outside computation on the forward pass were message ratios. That is, $w_e = \frac{m_{i\to\alpha}^{(t_{\max})}(\text{ON})}{m_{i\to\alpha}^{(t_{\max})}(\text{OFF})}$ for edge e corresponding $y_i = \text{ON}$. Thus, the final step is to backpropagate through this to update the adjoints of the messages. This is in contrast to all the other objectives functions considered in this thesis which are a function only of the beliefs.

Bibliography

Alfred V. Aho and Jeffrey D. Ullman (1972). *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc.

Bjoern Andres, Thorsten Beier, and Joerg H. Kappes (2012). “OpenGM: A C++ Library for Discrete Graphical Models”. In: *ArXiv e-prints*. arXiv: 1206.0111.

Michael Auli and Adam Lopez (2011). “A Comparison of Loopy Belief Propagation and Dual Decomposition for Integrated CCG Supertagging and Parsing”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Automatic Differentiation of Algorithms: Theory, Implementation, and Application (1991). SIAM.

Mohit Bansal, David Burkett, Gerard de Melo, and Dan Klein (2014). “Structured Learning for Taxonomy Induction with Belief Propagation”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio (2012). “Theano: new features and speed improvements”. In: *NIPS 2012 deep learning workshop*.

Amir Beck and Marc Teboulle (2003). “Mirror descent and nonlinear projected subgradient methods for convex optimization”. In: *Operations Research Letters* 31.3.

Yonatan Belinkov, Tao Lei, Regina Barzilay, and Amir Globerson (2014). “Exploring Compositional Architectures and Word Vector Representations for Prepositional Phrase Attachment”. In: *Transactions of the Association for Computational Linguistics (TACL)* 2.

Yoshua Bengio, Régis Cardin, Renato De Mori, and Yves Normandin (1990). “A hybrid coder for hidden Markov models using a recurrent neural networks”. In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Yoshua Bengio, Renato De Mori, Giovanni Flammia, and Ralf Kompe (1992). “Global optimization of a neural network-hidden Markov model hybrid”. In: *IEEE Transactions on Neural Networks* 3.2.

Yoshua Bengio and Paolo Frasconi (1995). “An input output HMM architecture”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle (2007). “Greedy Layer-Wise Training of Deep Networks”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Yoshua Bengio, Yann LeCun, Craig Nohl, and Chris Burges (1995). “LeRec: A NN/HMM hybrid for on-line handwriting recognition”. In: *Neural Computation* 7.6.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl (2011). “Algorithms for hyper-parameter optimization”. In: *Advances in Neural Information Processing Systems (NIPS)*.

James Bergstra and Yoshua Bengio (2012). “Random search for hyper-parameter optimization”. In: *Journal of Machine Learning Research (JMLR)* 13.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio (2010). “Theano: a CPU and GPU Math Expression Compiler”. In: *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

James Bergstra, Daniel Yamins, and David Cox (2013). “Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures”. In:

Peter J. Bickel and Kjell A. Doksum (1977). *Mathematical Statistics: Basic Ideas and Selected Topics*. Holden-Day Inc.

Anders Björkelund, Love Hafdell, and Pierre Nugues (2009). “Multilingual Semantic Role Labeling”. In: *Proceedings of the Computational Natural Language Learning (CoNLL) Shared Task*.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio (2012). “A semantic matching energy function for learning with multi-relational data”. In: *Machine Learning*.

Hervé Bourlard, Yochai Konig, and Nelson Morgan (1995). “REMAP: recursive estimation and maximization of a posteriori probabilities in connectionist speech recognition.” In: *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*.

Stephen Boxwell, Chris Brew, Jason Baldridge, Dennis Mehay, and Sujith Ravi (2011). “Semantic Role Labeling Without Treebanks?” In: *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*.

Stephen Boxwell and Michael White (2008). “Projecting Propbank Roles onto the CCGbank”. In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*.

Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai (1992). “Class-based n-gram models of natural language”. In: *Computational Linguistics* 18.4.

Sabine Buchholz and Erwin Marsi (2006). “CoNLL-X shared task on multilingual dependency parsing”. In: *Proceedings of Computational Natural Language Learning (CoNLL)*.

Razvan C. Bunescu and Raymond J. Mooney (2005). “A shortest path dependency kernel for relation extraction”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

David Burkett and Dan Klein (2012). “Fast Inference in Phrase Extraction Models with Belief Propagation”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Xavier Carreras (2007). “Experiments with a Higher-Order Projective Dependency Parser”. In: *Proceedings of the Computational Natural Language Learning (CoNLL) Shared Task*.

Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile (2004). “On the generalization ability of on-line learning algorithms”. In: *IEEE Transactions on Information Theory* 50.9.

Danqi Chen and Christopher Manning (2014). “A Fast and Accurate Dependency Parser using Neural Networks”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Massimiliano Ciaramita and Yasemin Altun (2006). “Broad-Coverage Sense Disambiguation and Information Extraction with a Supersense Sequence Tagger”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Shay Cohen and Noah A. Smith (2010). “Viterbi Training for PCFGs: Hardness Results and Competitiveness of Uniform Initialization”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater (2010). “Inducing tree-substitution grammars”. In: *Journal of Machine Learning Research (JMLR)* 9999.

Michael Collins (1999). “Head-driven statistical models for natural language parsing”. PhD thesis. University of Pennsylvania.

Michael Collins (2002). “Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

R. Collobert and J. Weston (2008). “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning”. In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Ronan Collobert (2011). “Deep learning for efficient discriminative parsing”. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet (2011a). “Torch7: A matlab-like environment for machine learning”. In: *Proceedings of Big Learning: The NIPS Workshop on Algorithms, Systems, and Tools for Learning at Scale*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa (2011b). “Natural language processing (almost) from scratch”. In: *Journal of Machine Learning Research (JMLR)* 12.

Aron Culotta and Jeffrey Sorensen (2004). “Dependency Tree Kernels for Relation Extraction”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Dipanjan Das, André F. T. Martins, and Noah A. Smith (2012). “An Exact Dual Decomposition Algorithm for Shallow Semantic Parsing with Constraints”. In: *Proceedings of the Joint Conference on Lexical and Computational Semantics (*SEM)*.

Trinh–Minh–Tri Do and Thierry Artieres (2010). “Neural conditional random fields”. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Justin Domke (2010). “Implicit Differentiation by Perturbation”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Justin Domke (2011). “Parameter Learning With Truncated Message-Passing”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Markus Dreyer and Jason Eisner (2009). "Graphical Models over Multiple Strings". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

John C. Duchi, Shai Shalev-Shwartz, Yoram Singer, and Ambuj Tewari (2010a). "Composite Objective Mirror Descent." In: *Proceedings of the Conference on Learning Theory (COLT)*.

John Duchi, Elad Hazan, and Yoram Singer (2010b). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Proceedings of the Conference on Learning Theory (COLT)*.

John Duchi, Elad Hazan, and Yoram Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research (JMLR)*.

John Duchi, Daniel Tarlow, Gal Elidan, and Daphne Koller (2006). "Using Combinatorial Optimization within Max-Product Belief Propagation". In: *Advances in Neural Information Processing Systems (NIPS)*.

Greg Durrett and Dan Klein (2015). "Neural CRF Parsing". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith (2015). "Transition-Based Dependency Parsing with Stack Long Short-Term Memory". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Chris Dyer, Jonathan Weese, Hendra Setiawan, Adam Lopez, Ferhan Ture, Vladimir Eidelman, Juri Ganitkevitch, Phil Blunsom, and Philip Resnik (2010). "cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models". In: *Proceedings of the Association for Computational Linguistics: System Demonstrations*.

Frederik Eaton and Zoubin Ghahramani (2009). "Choosing a variable to clamp". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Jason Eisner (1996). "Three New Probabilistic Models for Dependency Parsing: An Exploration". In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Jason Eisner and Noah A. Smith (2005). "Parsing with Soft and Hard Constraints on Dependency Length". In: *Proceedings of the International Workshop on Parsing Technologies (IWPT)*.

Gal Elidan, Ian Mcgraw, and Daphne Koller (2006). "Residual Belief Propagation: Informed Scheduling for Asynchronous Message Passing". In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Francis Ferraro, Max Thomas, Matthew R. Gormley, Travis Wolfe, Craig Harman, and Benjamin Van Durme (2014). "Concretely Annotated Corpora". In: *Proceedings of the Workshop on Automated Knowledge Base Construction (AKBC)*.

Nicholas FitzGerald, Oscar Täckström, Kuzman Ganchev, and Dipanjan Das (2015). "Semantic Role Labeling with Neural Network Factors". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

William Foland and James Martin (2015). "Dependency-Based Semantic Role Labeling using Convolutional Neural Networks". In:

Brendan J. Frey, Frank R. Kschischang, Hans-Andrea Loeliger, and Niclas Wiberg (1997). "Factor graphs and algorithms". In: *Proceedings of the Annual Allerton Conference on Communication Control and Computing*. Vol. 35.

Kuzman Ganchev and Mark Dredze (2008). "Small statistical models by random feature mixing". In: *Proceedings of the ACL08 HLT Workshop on Mobile Language Processing*.

Andrea Gesmundo, James Henderson, Paola Merlo, and Ivan Titov (2009). "A Latent Variable Model of Synchronous Syntactic-Semantic Parsing for Multiple Languages". In: *Proceedings of the Computational Natural Language Learning (CoNLL) Shared Task*.

Daniel Gildea and Daniel Jurafsky (2000). "Automatic Labeling of Semantic Roles". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Daniel Gildea and Daniel Jurafsky (2002). "Automatic labeling of semantic roles". In: *Computational Linguistics* 28 (3).

Daniel Gildea and Martha Palmer (2002). "The necessity of parsing for predicate argument recognition". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Joshua Goodman (1996). "Efficient Algorithms for Parsing the DOP Model". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Matthew R. Gormley (2015). *Pacaya—A Graphical Models and NLP Library*. Available from https://github.com/mgormley/pacaya.

Matthew R. Gormley, Mark Dredze, and Jason Eisner (2015a). "Approximation-aware Dependency Parsing by Belief Propagation". In: *Transactions of the Association for Computational Linguistics (TACL)*.

Matthew R. Gormley, Mark Dredze, Benjamin Van Durme, and Jason Eisner (2011). "Shared Components Topic Models with Application to Selectional Preference". In: *Proceedings of the NIPS Workshop on Learning Semantics*.

Matthew R. Gormley, Mark Dredze, Benjamin Van Durme, and Jason Eisner (2012). "Shared Components Topic Models". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Matthew R. Gormley and Jason Eisner (2013). "Nonconvex Global Optimization for Grammar Induction". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Matthew R. Gormley and Jason Eisner (2014). "Structured Belief Propagation for NLP". In: *Proceedings of the Association for Computational Linguistics: Tutorials*.

Matthew R. Gormley and Jason Eisner (2015). "Structured Belief Propagation for NLP". In: *Proceedings of the Association for Computational Linguistics: Tutorials*.

Matthew R. Gormley, Adam Gerber, Mary Harper, and Mark Dredze (2010). "Non-Expert Correction of Automatically Generated Relation Annotations". In: *Proceedings of the NAACL Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*.

Matthew R. Gormley, Margaret Mitchell, Benjamin Van Durme, and Mark Dredze (2014). "Low-Resource Semantic Role Labeling". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Matthew R. Gormley, Mo Yu, and Mark Dredze (2015b). "Improved Relation Extraction with Feature-rich Compositional Embedding Models". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Spence Green, Nicholas Andrews, Matthew R. Gormley, Mark Dredze, and Christopher D. Manning (2012). “Entity Clustering Across Languages”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Patrick Haffner (1993). “Connectionist speech recognition with a global MMI algorithm”. In: *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*.

Aria Haghighi and Dan Klein (2006). “Prototype-Driven Learning for Sequence Models”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang (2009). “The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages”. In: *Proceedings of the Computational Natural Language Learning (CoNLL) Shared Task*.

Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka (2015). “Task-Oriented Learning of Word Embeddings for Semantic Relation Classification”. In: *ArXiv e-prints*.

Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz (2010). “SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations Between Pairs of Nominals”. In: *Proceedings of the International Workshop on Semantic Evaluation (SemEval)*.

Karl Moritz Hermann and Phil Blunsom (2013). “The role of syntax in vector space models of compositional semantics”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Karl Moritz Hermann, Dipanjan Das, Jason Weston, and Kuzman Ganchev (2014). “Semantic Frame Identification with Distributed Word Representations”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

G. E Hinton (2002). “Training products of experts by minimizing contrastive divergence”. In: *Neural Computation* 14.8.

Julia Hockenmaier and Mark Steedman (2007). “CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank”. In: *Computational Linguistics* 33.3.

Liang Huang, Suphan Fayong, and Yang Guo (2012). “Structured Perceptron with Inexact Search”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell (2014). “Caffe: Convolutional architecture for fast feature embedding”. In: *Proceedings of the ACM International Conference on Multimedia*.

Jing Jiang and ChengXiang Zhai (2007). “A Systematic Exploration of the Feature Space for Relation Extraction”. In:

Richard Johansson (2009). “Statistical Bistratal Dependency Parsing”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Richard Johansson and Pierre Nugues (2008). “Dependency-based Semantic Role Labeling of PropBank”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Dan Klein and Christopher Manning (2004). “Corpus-Based Induction of Syntactic Structure: Models of Dependency and Constituency”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Dan Klein and Christopher D. Manning (2001). “Parsing and Hypergraphs”. In: *Proceedings of the International Workshop on Parsing Technologies (IWPT)*.

Terry Koo, Xavier Carreras, and Michael Collins (2008). “Simple Semi-supervised Dependency Parsing”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Terry Koo and Michael Collins (2010). “Efficient third-order dependency parsers”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag (2010). “Dual decomposition for parsing with non-projective head automata”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger (2001). “Factor graphs and the sum-product algorithm”. In: *IEEE Transactions on Information Theory* 47.2.

Alex Kulesza and Fernando Pereira (2008). “Structured Learning with Approximate Inference.” In: *Advances in Neural Information Processing Systems (NIPS)*.

John Lafferty, Andrew McCallum, and Fernando Pereira (2001). “Conditional random fields: Probabilistic models for segmenting and labeling sequence data”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Vol. 1.

Qi Li and Heng Ji (2014). “Incremental Joint Extraction of Entity Mentions and Relations”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Zhifei Li and Jason Eisner (2009). “First- and Second-Order Expectation Semirings with Applications to Minimum-Risk Training on Translation Forests”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Percy Liang (2005). “Semi-supervised learning for natural language”. PhD thesis. Massachusetts Institute of Technology.

Yang Liu, Furu Wei, Sujian Li, Heng Ji, Ming Zhou, and Houfeng WANG (2015). “A Dependency-Based Neural Network for Relation Classification”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Xavier Lluís, Xavier Carreras, and Lluís Màrquez (2013). “Joint Arc-factored Parsing of Syntactic and Semantic Dependencies”. In: *Transactions of the Association for Computational Linguistics (TACL)*.

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams (2015). “Gradient-based Hyperparameter Optimization through Reversible Learning”. In: *ArXiv e-prints*. arXiv: 1502.03492.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky (2014). “The Stanford CoreNLP Natural Language Processing Toolkit”. In: *Proceedings of the Association for Computational Linguistics: System Demonstrations*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini (1993). "Building a large annotated corpus of English: The Penn Treebank". In: *Computational Linguistics* 19.2.

André F. T. Martins, Kevin Gimpel, Noah A. Smith, Eric P. Xing, Mario A. T. Figueiredo, and Pedro M. Q. Aguiar (2010a). *Learning structured classifiers with dual coordinate ascent*. Tech. rep. CMU-ML-10-109. Carnegie Mellon University.

André F. T. Martins, Noah A. Smith, Pedro M. Q. Aguiar, and Mario A. T. Figueiredo (2011a). "Structured Sparsity in Structured Prediction". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

André F. T. Martins, Noah A. Smith, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo (2011b). "Dual decomposition with many overlapping components". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

André F. T. Martins, Noah A. Smith, and Eric P. Xing (2009). "Concise Integer Linear Programming Formulations for Dependency Parsing". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

André F.T. Martins, Miguel B. Almeida, and Noah A. Smith (2013). "Turning on the turbo: Fast third-order non-projective turbo parsers". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

André F.T. Martins, Noah A. Smith, Eric P. Xing, Pedro M. Q. Aguiar, and Mario A. T. Figueiredo (2010b). "Turbo Parsers: Dependency Parsing by Approximate Variational Inference". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Andrew McCallum, Karl Schultz, and Sameer Singh (2009). "FACTORIE: Probabilistic Programming via Imperatively Defined Factor Graphs". In: *Advances in Neural Information Processing Systems (NIPS)*.

Ryan McDonald, Koby Crammer, and Fernando Pereira (2005). "Online large-margin training of dependency parsers". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Ryan McDonald and Fernando Pereira (2006). "Online Learning of Approximate Dependency Parsing Algorithms". In: *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "Distributed representations of words and phrases and their compositionality". In: *ArXiv e-prints*.

Scott Miller, Jethran Guinness, and Alex Zamanian (2004). "Name Tagging with Word Clusters and Discriminative Training". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

T. Minka, J.M. Winn, J.P. Guiver, and D.A. Knowles (2012). *Infer.NET 2.5*. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

Alexis Mitchell, Stephanie Strassel, Shudong Huang, and Ramez Zakhary (2005). "Ace 2004 multilingual training corpus". In: *Linguistic Data Consortium*.

Andriy Mnih and Geoffrey Hinton (2007). "Three new graphical models for statistical language modelling". In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Joris M. Mooij (2010). "libDAI: A Free and Open Source C++ Library for Discrete Approximate Inference in Graphical Models". In: *Journal of Machine Learning Research (JMLR)* 11.

Frederic Morin and Yoshua Bengio (2005). "Hierarchical probabilistic neural network language model". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Kevin P. Murphy, Yair Weiss, and Michael I. Jordan (1999). "Loopy belief propagation for approximate inference: An empirical study". In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Courtney Napoles, Matthew R. Gormley, and Benjamin Van Durme (2012). "Annotated Gigaword". In: *Proceedings of the Workshop on Automated Knowledge Base Construction (AKBC)*.

Jason Naradowsky, Sebastian Riedel, and David Smith (2012a). "Improving NLP through Marginalization of Hidden Syntactic Structure". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Jason Naradowsky, Tim Vieira, and David A. Smith (2012b). "Grammarless Parsing for Joint Inference". In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Tahira Naseem and Regina Barzilay (2011). "Using Semantic Cues to Learn Syntax." In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Tahira Naseem, Harr Chen, Regina Barzilay, and Mark Johnson (2010). "Using Universal Linguistic Knowledge to Guide Grammar Induction". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Arkadi Nemirovsky and David Yudin (1983). *Problem complexity and method efficiency in optimization.* Wiley.

Thien Huu Nguyen and Ralph Grishman (2014). "Employing Word Representations and Regularization for Domain Adaptation of Relation Extraction". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Thien Huu Nguyen and Ralph Grishman (2015). "Relation Extraction: Perspective from Convolutional Neural Networks". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Thien Huu Nguyen, Barbara Plank, and Ralph Grishman (2015). "Semantic Representations for Domain Adaptation: A Case Study on the Tree Kernel-based Method for Relation Extraction". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Feng Ning, Damien Delhomme, Yann LeCun, Fabio Piano, Léon Bottou, and Paolo Emilio Barbano (2005). "Toward automatic phenotyping of developing embryos from videos". In: *IEEE Transactions on Image Processing* 14.9.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret (2007). "The CoNLL 2007 Shared Task on Dependency Parsing". In: *Proceedings of the Computational Natural Language Learning (CoNLL) Shared Task*.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda (2011). "English Gigaword Fifth Edition, June". In: *Linguistic Data Consortium*.

Adam Pauls and Dan Klein (2009). "Hierarchical search for parsing". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

J. Pearl (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Wenzhe Pei, Tao Ge, and Baobao Chang (2015). "An Effective Neural Network Model for Graph-based Dependency Parsing". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Nanyun Peng, Francis Ferraro, Mo Yu, Nicholas Andrews, Jay DeYoung, Max Thomas, Matthew R. Gormley, Travis Wolfe, Craig Harman, Benjamin Van Durme, and Mark Dredze (2015). "A Concrete Chinese NLP Pipeline". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL): Demonstration Session*.

Fernando Pereira and Yves Schabes (1992). "Inside-Outside Reestimation from Partially Bracketed Corpora". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Slav Petrov (2009). "Coarse-to-Fine Natural Language Processing". PhD thesis. University of California at Bekeley.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein (2006). "Learning Accurate, Compact, and Interpretable Tree Annotation". In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Slav Petrov, Dipanjan Das, and Ryan McDonald (2012). "A Universal Part-of-Speech Tagset". In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*.

Slav Petrov, Aria Haghighi, and Dan Klein (2008). "Coarse-to-fine syntactic machine translation using language projections". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Slav Petrov and Dan Klein (2007). "Improved Inference for Unlexicalized Parsing". In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Nugues Pierre and Kalep Heiki-Jaan (2007). "Extended constituent-to-dependency conversion for English". In: *NODALIDA 2007 Proceedings*.

Barbara Plank and Alessandro Moschitti (2013). "Embedding Semantic Similarity in Tree Kernels for Domain Adaptation of Relation Extraction". In: *Proceedings of the Association for Computational Linguistics (ACL)*.

V. Punyakanok, D. Roth, and W. Yih (2005). "The Necessity of Syntactic Parsing for Semantic Role Labeling". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih (2008). "The Importance of Syntactic Parsing and Inference in Semantic Role Labeling". In: *Computational Linguistics* 34.2.

Rami Al-Rfou', Bryan Perozzi, and Steven Skiena (2013). "Polyglot: Distributed Word Representations for Multilingual NLP". In: *Proceedings of Computational Natural Language Learning (CoNLL)*.

Sebastian Riedel and James Clarke (2006). "Incremental integer linear programming for non-projective dependency parsing". In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Sebastian Riedel and David A. Smith (2010). “Relaxed Marginal Inference and its Application to Dependency Parsing”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Bryan Rink and Sanda Harabagiu (2010). “UTD: Classifying Semantic Relations by Combining Lexical and Semantic Resources”. In: *Proceedings of the International Workshop on Semantic Evaluation (SemEval)*.

Stephane Ross, Daniel Munoz, Martial Hebert, and J. Andrew Bagnell (2011). “Learning message-passing inference machines for structured prediction”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Michael Roth and Kristian Woodsend (2014). “Composition of Word Representations Improves Semantic Role Labelling”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams (1986). “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1. MIT Press.

Alexander M. Rush and Slav Petrov (2012). “Vine pruning for efficient multi-pass dependency parsing”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Cicero Nogueira dos Santos, Bing Xiang, and Bowen Zhou (2015). “Classifying Relations by Ranking with Convolutional Neural Networks”. In: *ArXiv e-prints*.

David A. Smith and Jason Eisner (2006). “Minimum-Risk Annealing for Training Log-Linear Models”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

David A. Smith and Jason Eisner (2008). “Dependency Parsing by Belief Propagation”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

N.A. Smith (2006). “Novel estimation methods for unsupervised discovery of latent structure in natural language text”. PhD thesis. Johns Hopkins University.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams (2012). “Practical Bayesian optimization of machine learning algorithms”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Justin Snyder, Rebecca Knowles, Mark Dredze, Matthew R. Gormley, and Travis Wolfe (2013). “Topic Models and Metadata for Visualizing Text Corpora”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL): Demonstration Session*.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng (2013a). “Parsing with compositional vector grammars”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng (2012). “Semantic Compositionality through Recursive Matrix-Vector Spaces”. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts (2013b). “Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Anders Søgaard (2012). “Two baselines for unsupervised dependency parsing”. In: *Proceedings of the NAACL-HLT Workshop on the Induction of Linguistic Structure*.

Valentin I. Spitkovsky, Hiyan Alshawi, Angel X. Chang, and Daniel Jurafsky (2011). “Unsupervised Dependency Parsing without Gold Part-of-Speech Tags”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky (2013). “Breaking Out of Local Optima with Count Transforms and Model Recombination: A Study in Grammar Induction”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Valentin I. Spitkovsky, Hiyan Alshawi, Daniel Jurafsky, and Christopher D Manning (2010a). “Viterbi Training Improves Unsupervised Dependency Parsing”. In: *Proceedings of Computational Natural Language Learning (CoNLL)*.

Valentin I. Spitkovsky, Daniel Jurafsky, and Hiyan Alshawi (2010b). “Profiting from Mark-Up: Hyper-Text Annotations for Guided Parsing”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Valentin Ilyich Spitkovsky (2013). “Grammar Induction and Parsing with Dependency-and-Boundary Models”. PhD thesis. Computer Science Department, Stanford University.

Vivek Srikumar and Christopher D Manning (2014). “Learning Distributed Representations for Structured Output Prediction”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Veselin Stoyanov and Jason Eisner (2012). “Minimum-Risk Training of Approximate CRF-Based NLP Systems”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner (2011). “Empirical Risk Minimization of Graphical Model Parameters Given Approximate Inference, Decoding, and Model Structure”. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Ang Sun, Ralph Grishman, and Satoshi Sekine (2011). “Semi-supervised Relation Extraction with Large-scale Word Clustering”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre (2008). “The CoNLL- 2008 shared task on joint parsing of syntactic and semantic dependencies”. In: *Proceedings of Computational Natural Language Learning (CoNLL)*.

Charles Sutton and Andrew McCallum (2007). “An Introduction to Conditional Random Fields for Relational Learning”. In: *Introduction to Statistical Relational Learning*. MIT Press.

Jonathan J Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler (2014). “Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Kristina Toutanova, Aria Haghighi, and Christopher Manning (2005). “Joint Learning Improves Semantic Role Labeling”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio (2010). “Word representations: a simple and general method for semi-supervised learning”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton (2015). “Grammar as a Foreign Language”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Martin J. Wainwright (2006). “Estimating the “wrong” graphical model: Benefits in the computation-limited setting”. In: *Journal of Machine Learning Research (JMLR)* 7.

Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda (2006). “ACE 2005 multilingual training corpus”. In: *Linguistic Data Consortium*.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg (2009). “Feature hashing for large scale multitask learning”. In: *Proceedings of the International Conference on Machine Learning (ICML)*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov (2015). “Structured Training for Neural Network Transition-Based Parsing”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

David Weiss and Ben Taskar (2010). “Structured prediction cascades”. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Lin Xiao (2009). “Dual averaging method for regularized stochastic learning and online optimization”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Hiroyasu Yamada and Yuji Matsumoto (2003). “Statistical dependency analysis with support vector machines”. In: *Proceedings of the International Workshop on Parsing Technologies (IWPT)*. Vol. 3.

Jonathan S Yedidia, William T. Freeman, and Yair Weiss (2000). “Generalized Belief Propagation”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Daniel H. Younger (1967). “Recognition and parsing of context-free languages in time n^3”. In: *Information and Control* 10.2.

Mo Yu (2015). “Modeling and Learning of Distributed Representations for Natural Language Structures”. PhD thesis. Harbin Institute of Technology.

Mo Yu and Mark Dredze (2015). “Learning Composition Models for Phrase Embeddings”. In: *Transactions of the Association for Computational Linguistics (TACL)* 3.

Mo Yu, Matthew R. Gormley, and Mark Dredze (2014). “Factor-based Compositional Embedding Models”. In: *Proceedings of the NIPS Workshop on Learning Semantics*.

Mo Yu, Matthew R. Gormley, and Mark Dredze (2015). “Combining Word Embeddings and Feature Embeddings for Fine-grained Relation Extraction”. In: *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella (2003). “Kernel methods for relation extraction”. In: *Journal of Machine Learning Research (JMLR)* 3.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao (2014). “Relation Classification via Convolutional Deep Neural Network”. In: *Proceedings of the International Conference on Computational Linguistics (COLING)*.

Hao Zhang, Liang Huang, Kai Zhao, and Ryan McDonald (2013). “Online Learning for Inexact Hypergraph Search”. In: *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Hai Zhao, Wenliang Chen, Chunyu Kity, and Guodong Zhou (2009). “Multilingual Dependency Learning: A Huge Feature Engineering Method to Semantic Dependency Parsing”. In: *Proceedings of the Computational Natural Language Learning (CoNLL) Shared Task*.

GuoDong Zhou, Jian Su, Jie Zhang, and Min Zhang (2005). “Exploring Various Knowledge in Relation Extraction”. In: *Proceedings of the Association for Computational Linguistics (ACL)*.

Vita

Matt Gormley graduated from Schenley High School in Pittsburgh with an International Baccalaureate diploma. He obtained a Bachelor's degree in Computer Science with a double major in Cognitive Science from Carnegie Mellon University. During his studies he interned with the Speech and Natural Language group at Microsoft. After receiving a *Certificado de Cocina Española* from *El Centro Superior de Hostelería y Turismo de Valencia*, he worked at Endeca Technologies, Inc. He obtained his M.S.Eng. and Ph.D. in Computer Science at Johns Hopkins University, where he was co-advised by Mark Dredze and Jason Eisner. At Hopkins, his research was funded by a Human Language Technology Center of Excellence fellowship and a Fred Jelinek Fellowship. During his Ph.D., he interned at Google. In 2016, Matt is returning to Carnegie Mellon University to join the faculty of the Machine Learning department as an assistant teaching professor.