Approximation-Aware Dependency Parsing by Belief Propagation

Matthew R. Gormley **Mark Dredze** **Jason Eisner**
Human Language Technology Center of Excellence
Center for Language and Speech Processing
Department of Computer Science
Johns Hopkins University, Baltimore, MD
{mrg,mdredze,jason}@cs.jhu.edu

Abstract

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 backpropagation. 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 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 paper, 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*.[1]

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."

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

[1] 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).

Transactions of the Association for Computational Linguistics, vol. 3, pp. 489–501, 2015. Action Editor: Sebastian Riedel.
Submission batch: 4/2015; Published 8/2015.

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. 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.

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) 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 1 for an example factor graph.

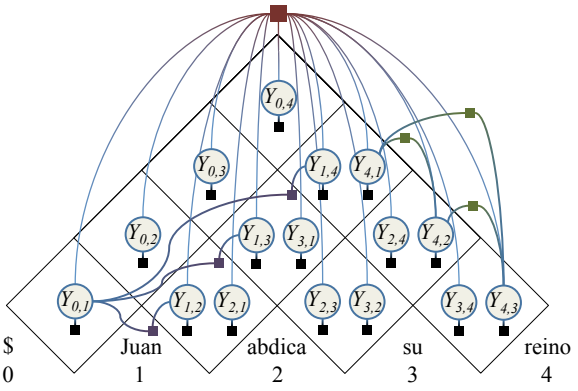

Figure 1: Factor graph for dependency parsing of a 4-word sentence; $ is the root of the dependency graph. The boolean variable $Y_{h,m}$ encodes whether the edge from parent h to child m is present. The unary factor (black) connected to this variable scores the edge in isolation (given the sentence). 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 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 $\boldsymbol{y}^*$:

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}^*) = \textstyle\sum_{i:\, \hat{y}_i = \text{OFF}} \mathbb{I}(y_i^* = \text{ON}) \qquad (1)$$

$\mathbb{I}$ 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, 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:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \operatorname*{argmin}_{\hat{\boldsymbol{y}}} \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})] \quad (2)$$

$$= \operatorname*{argmax}_{\hat{\boldsymbol{y}}} \sum_{i:\, \hat{y}_i = \text{ON}} p_{\boldsymbol{\theta}}(y_i = \text{ON} \mid \boldsymbol{x}) \quad (3)$$

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.[2]

All systems in this paper 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 (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 paper proposes a training procedure that 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 (3) by running a first-order dependency parser with edge scores equal to those probabilities.[3] 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 (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 (3), as well as the factor marginals $p_{\boldsymbol{\theta}}(\boldsymbol{y}_\alpha \mid \boldsymbol{x}) = \sum_{\boldsymbol{y}': \boldsymbol{y}'_\alpha = \boldsymbol{y}_\alpha} p_{\boldsymbol{\theta}}(\boldsymbol{y}' \mid \boldsymbol{x})$. The algorithm proceeds by iteratively sending messages from variables, y_i, to factors, α:

$$m^{(t)}_{i \to \alpha}(y_i) \propto \prod_{\beta \in \mathcal{N}(i) \setminus \alpha} m^{(t-1)}_{\beta \to i}(y_i) \quad (4)$$

and from factors to variables:

$$m^{(t)}_{\alpha \to i}(y_i) \propto \sum_{\boldsymbol{y}_\alpha \sim y_i} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} m^{(t-1)}_{j \to \alpha}(y_i) \quad (5)$$

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:

$$b_i(y_i) \propto \prod_{\alpha \in \mathcal{N}(i)} m^{(t_{\text{max}})}_{\alpha \to i}(y_i) \quad (6)$$

$$b_\alpha(\boldsymbol{y}_\alpha) \propto \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{i \in \mathcal{N}(\alpha)} m^{(t_{\text{max}})}_{i \to \alpha}(y_i) \quad (7)$$

We assume each of the messages and beliefs given in (4)–(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.[4]

For the PTREE factor, the summation over variable assignments required for $m^{(t)}_{\alpha \to i}(y_i)$ in Eq. (5) equates to a summation over exponentially many projective parse trees. However, we can use an inside-outside variant of Eisner (1996)'s algorithm

[2] If we used a simple 0-1 loss function within (2), then MBR decoding would reduce to MAP decoding.

[3] 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.

[4] 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.

to compute this in polynomial time (we describe this as hypergraph parsing in §3). The resulting "structured BP" inference procedure—detailed by Smith and Eisner (2008)—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.[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 $\{(\boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)})\}_{d=1}^{D}$.

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{D}\Big(\big(\sum_{d=1}^{D} J(\boldsymbol{\theta}; \boldsymbol{x}^{(d)}, \boldsymbol{y}^{(d)}) \big) + \frac{\lambda}{2} ||\boldsymbol{\theta}||_2^2 \Big) \tag{8}$$

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.[6]

Objective Functions The standard choice for J is the negative conditional log-likelihood (§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):

$$\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} \mathbb{E}[\ell(h_{\boldsymbol{\theta}}(X), Y)] \tag{9}$$

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 (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).

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 (Griewank and Corliss, 1991). This yields the same type of "backpropagation" 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 Feedfoward 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 feedfoward 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 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

[5] 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.

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

how we define the function J (the forward pass) and how we compute its gradient (the backward pass). Backpropagation through the circuit from Figure 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 (§5.2). The remaining challenges form the primary technical contribution of this paper:

1. Our true loss function $\ell(h_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}^*)$ by way of the decoder $h_{\boldsymbol{\theta}}$ contains an argmax (3) over trees and is therefore not differentiable. We show how to soften this decoder (by substituting a softmax), making it differentiable (§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 (§4.2).
3. We show how to run backprop through the inside-outside algorithm on a hypergraph (§5.4) for use in two modules: the softened decoder (§5.1) and computation of messages from the PTREE factor (§5.3). This allows us to go beyond Stoyanov et al. (2011) and train *structured* BP in an approximation-aware and loss-aware fashion.

4 Differentiable Objective Functions

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 $p_{\boldsymbol{\theta}}(y_i = \textsc{on} \mid \boldsymbol{x})$ as input, or in practice, their BP approximations $b_i(\textsc{on})$:

$$q^{1/T}_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}} \mid \boldsymbol{x}) \propto \exp\left(\textstyle\sum_{i:\hat{y}_i=\textsc{on}} \frac{p_{\boldsymbol{\theta}}(y_i=\textsc{on} \mid \boldsymbol{x})}{T}\right) \quad (10)$$

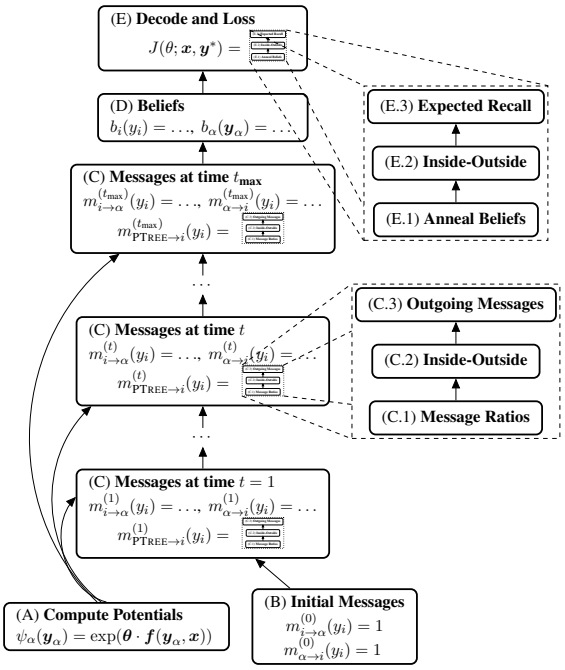

Figure 2: Feed-forward topology of inference, decoding, and loss. (E.1–E.3) show the *annealed risk*, one of the objective functions we consider.

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:

$$R^{1/T}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}^*) = \mathbb{E}_{\hat{\boldsymbol{y}} \sim q^{1/T}_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}^*)] \quad (11)$$

As $T \to 0$ ("annealing"), the decoder almost always chooses the MBR parse,[7] 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 $R^{1/T}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}^*) = -\sum_{i:y^*_i=\textsc{on}} q^{1/T}_{\boldsymbol{\theta}}(\hat{y}_i = \textsc{on} \mid \boldsymbol{x})$. This is the negated **expected recall** of a parse $\hat{\boldsymbol{y}} \sim q^{1/T}_{\boldsymbol{\theta}}$. We obtain the required marginals $q^{1/T}_{\boldsymbol{\theta}}(\hat{y}_i = \textsc{on} \mid \boldsymbol{x})$ from (10) by running inside-

[7] Recall from (3) that the MBR parse is the tree $\hat{\boldsymbol{y}}$ that maximizes the sum $\sum_{i:\hat{y}_i=\textsc{on}} p_{\boldsymbol{\theta}}(y_i = \textsc{on} \mid \boldsymbol{x})$. As $T \to 0$, the right-hand side of (10) grows fastest for this $\hat{\boldsymbol{y}}$, so its probability under $q^{1/T}_{\boldsymbol{\theta}}$ approaches 1 (or $1/k$ if there is a k-way tie for MBR parse).

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 (§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.

4.2 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. 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 $\mathbf{L_2}$ **distance** loss function:

$$J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}^*) = \sum_i \sum_{y_i} (b_i(y_i) - b_i^*(y_i))^2 \qquad (12)$$

We can use this L_2 distance objective function for training, adding the MBR decoder and loss evaluation back in only at test time.

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. 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 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 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 gives a summary of the adjoint computations we require. Due to space constraints, we direct the reader to the extended version of this paper (Gormley et al., 2015a) for full details of all the adjoints.

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 2 (E). Consider the simple case where J is *L_2 distance* from (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 (11), we compute $\eth b_i(y_i)$ by applying backpropagation recursively to our algorithm for J from §4.1. This sub-algorithm defines a subcircuit depicted in Figure 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 §5.4.

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 (4)–(7). Backpropagation differentiates these elementary expressions. First, using the belief adjoints, we compute the adjoints of the final messages ($\eth m^{(t_{\max})}_{j\to\alpha}(y_j)$, $\eth m^{(t_{\max})}_{\beta\to i}(y_i)$) by applying the chain rule to Eqs. (6) and (7). This is the backward pass through Fig. 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. (4) and (5). Backprop works in the opposite order, updating the adjoints of the messages at time $t-1$ and the potential functions ($\eth m^{(t-1)}_{j\to\alpha}(y_j)$, $\eth m^{(t-1)}_{\beta\to i}(y_i)$, $\eth \psi_\alpha(\boldsymbol{y}_\alpha)$) 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. 2 (C). The backward pass through Fig. 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. 2 (A). For the explicit formula of these adjoints, see Gormley et al. (2015a) or Appendix A.1 of Stoyanov et al. (2011). The next section handles the special case of $\eth m^{(t)}_{j\to\text{PTREE}}(y_j)$.

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), which computes the message from a factor α to a variable y_i, in the special case where $\alpha = \text{PTREE}$. They 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 $\prod_i m^{(t)}_{i\to\alpha}(\text{OFF})$ to get the marginal factor belief $b_\alpha(y_i)$. Finally they divide the belief by the incoming message $m^{(t)}_{i\to\alpha}(\text{ON})$ to get the corresponding outgoing message $m^{(t+1)}_{\alpha\to i}(\text{ON})$. These steps are shown in Figure 2 (C.1–C.3), and are repeated each time we send a message from the PTree factor.

Similarly, we exploit the structure of this algorithm to compute the adjoints $\eth m^{(t)}_{j\to\text{PTREE}}(y_j)$. 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.

5.4 Backprop of Hypergraph Inside-Outside

Both the annealed risk loss function (§4.1) and the computation of messages from the PTREE factor (§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 $\alpha_{\text{root}} = 1$.

$$\beta_i = \sum_{e \in I(i)} w_e \prod_{j \in T(e)} \beta_j \tag{13}$$

$$\alpha_j = \sum_{e \in O(i)} w_e \, \alpha_{H(e)} \prod_{j \in T(e): j \neq i} \beta_j \tag{14}$$

$$p_w(i) = \alpha_i \beta_i / \beta_{\text{root}} \tag{15}$$

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.

$$\eth \alpha_i = \eth p_w(i) \tfrac{\partial p_w(i)}{\partial \alpha_i} + \sum_{e \in I(i)} \sum_{j \in T(e)} \eth \alpha_j \tfrac{\partial \alpha_j}{\partial \alpha_i} \tag{16}$$

$$\eth \beta_{\text{root}} = \sum_{i \neq \text{root}} \eth p_w(i) \tfrac{\partial p_w(i)}{\partial \beta_{\text{root}}} \tag{17}$$

$$\eth\beta_j = \eth p_w(j)\frac{\partial p_w(j)}{\partial \beta_j} + \sum_{e \in O(j)} \eth\beta_{H(e)}\frac{\partial \beta_{H(e)}}{\partial \beta_j} + \sum_{e \in O(j)} \sum_{k \in T(e): k \neq j} \eth\alpha_k \frac{\partial \alpha_k}{\partial \beta_j} \quad \forall j \neq \text{root} \quad (18)$$

$$\eth w_e = \eth\beta_{H(e)}\frac{\partial \beta_{H(e)}}{\partial w_e} + \sum_{j \in T(e)} \eth\alpha_j \frac{\partial \alpha_j}{\partial w_e} \quad (19)$$

The partial derivatives required for the above adjoints are given in the extended version of this paper (Gormley et al., 2015a). This backpropagation method is used for both Figure 2 (C.2) and (E.2).

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 (§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 2, we can use a standard dynamic programming algorithm for dependency parsing, which is simply another instance of inside-outside on a hypergraph (§5.4). The exact marginals from inside-outside (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: $J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}^*) = -\log p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$. 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). 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).

7 Experiments

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 (§2). We add $O(n^3)$ second-order grandparent and arbitrary-sibling factors as in Riedel and Smith (2010) and Martins et al. (2010). 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).[8] 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 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

[8] 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.

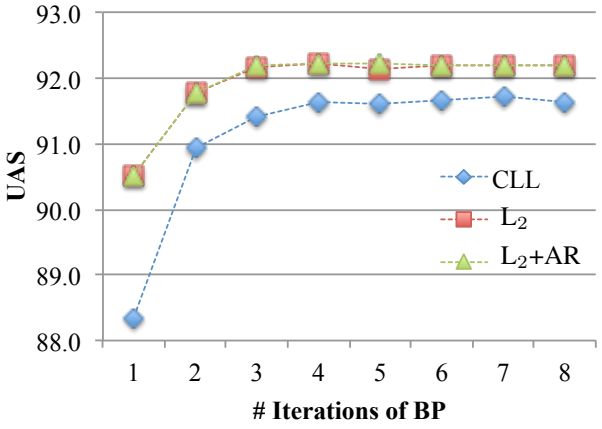

Figure 3: Speed/accuracy tradeoff of English PTB-YM UAS vs. the *total* number of BP iterations $t_{\max}$ for standard conditional likelihood training (CLL) and our approximation-aware training with either an L_2 objective (L_2) or a staged training of L_2 followed by annealed risk (L_2+AR). Note that the x-axis shows the number of iterations used for *both* training and testing. We use a 2nd-order model with Grand.+Sib. factors.

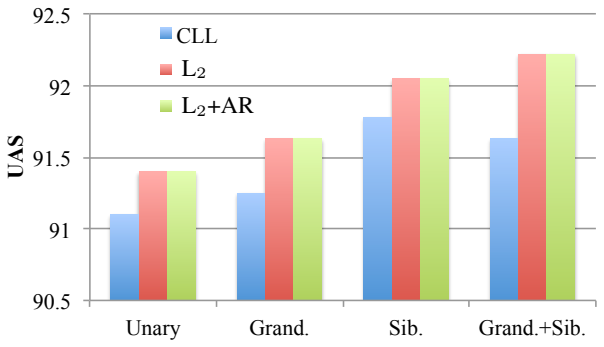

Figure 4: English PTB-YM UAS vs. the types of 2nd-order factors included in the model for approximation-aware training and standard conditional likelihood training. All models include 1st-order factors (Unary). The 2nd-order models include grandparents (Grand.), arbitrary siblings (Sib.), or both (Grand.+Sib.)—and use 4 iterations of BP.

POS tags for the CoNLL languages, and predicted tags from TurboTagger (Martins et al., 2013) for the PTB. Unlike most prior work, we hold out 10% of each CoNLL training dataset as development data for regularization by early stopping.[9]

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 projective tree (Carreras, 2007).[10] 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.

Learning Settings We compare three learning settings. The first, our baseline, is conditional log-likelihood training (CLL) (§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 (§3) with either our L_2 distance objective (L_2) (§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 (§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 (§6). The ℓ_2-regularizer weight in (8) is $\lambda = 1$. Each method is trained by AdaGrad for 5 epochs with early stopping (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.

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

[9] 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.

[10] 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.

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 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 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.

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.

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 1: 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.

Exact Inference with Grandparents §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 1 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 2). 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. 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 that larger training sets would also favor our approach, by providing a clearer signal of how to reduce the objective (8).

	1ST-ORDER		2ND-ORDER (WITH GIVEN NUM. BP ITERATIONS)							
			1		2		4		8	
LANGUAGE	CLL	$L_2 - CLL$	CLL	$L_2 - CLL$	CLL	$L_2 - CLL$	CLL	$L_2 - CLL$	CLL	$L_2 - CLL$
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 2: Results on 19 languages from CoNLL-2006/2007. For languages appearing in both datasets, the 2006 version was used, except for Chinese (ZH). Evaluation follows the 2006 conventions and excludes punctuation. We report *absolute* UAS for the baseline (CLL) and the *improvement* in UAS for L_2 over CLL ($L_2 - CLL$) with positive/negative differences in blue/red. The average UAS and average difference across all languages (AVG.) is given.

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 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., 2012), TAG (Auli and Lopez, 2011), ITG-constraints for phrase extraction (Burkett and Klein, 2012), and graphical models over strings (Dreyer and Eisner, 2009).

9 Conclusions

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. For additional details, see the tech report version of this paper (Gormley et al., 2015a). Our code is available in a general-purpose library for structured BP, hypergraphs, and backprop (Gormley, 2015).

Acknowledgments This research was funded by the Human Language Technology Center of Excellence at Johns Hopkins University. Thanks to the anonymous reviewers for their insightful comments.

References

Michael Auli and Adam Lopez. 2011. A comparison of loopy belief propagation and dual decomposition for integrated CCG supertagging and parsing. In *Proceedings of ACL*.

Mohit Bansal, David Burkett, Gerard de Melo, and Dan Klein. 2014. Structured learning for taxonomy induction with belief propagation. In *Proceedings of ACL*.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. In B. Schölkopf, J.C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*.

Peter J. Bickel and Kjell A. Doksum. 1977. *Mathematical Statistics: Basic Ideas and Selected Topics*. Holden-Day Inc., Oakland, CA, USA.

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

David Burkett and Dan Klein. 2012. Fast inference in phrase extraction models with belief propagation. In *Proceedings of NAACL-HLT*.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*.

Justin Domke. 2010. Implicit differentiation by perturbation. In *Advances in Neural Information Processing Systems*.

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 EMNLP*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*.

Greg Durrett and Dan Klein. 2015. Neural CRF Parsing. In *Proceedings of ACL*.

Frederik Eaton and Zoubin Ghahramani. 2009. Choosing a variable to clamp. In *Proceedings of AISTATS*.

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)*.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING*.

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*, volume 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*.

Joshua Goodman. 1996. Efficient algorithms for parsing the DOP model. In *Proceedings of EMNLP*.

Matthew R. Gormley, Mark Dredze, and Jason Eisner. 2015a. Approximation-aware dependency parsing by belief propagation (extended version). Technical report available from `arXiv.org` as `arXiv:1508.02375`.

Matthew R. Gormley, Mo Yu, and Mark Dredze. 2015b. Improved relation extraction with feature-rich compositional embedding models. In *Proceedings of EMNLP*.

Matthew R. Gormley. 2015. Pacaya—a graphical models and NLP library. Available from `https://github.com/mgormley/pacaya`.

Andreas Griewank and George F. Corliss, editors. 1991. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. SIAM, Philadelphia, PA.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of NAACL-HLT*.

Dan Klein and Christopher D. Manning. 2001. Parsing and hypergraphs. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL*.

Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. 2001. Factor graphs and the sum-product algorithm. *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*.

Zhifei Li and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of EMNLP*.

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

André F. T. Martins, Noah A. Smith, and Eric P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of ACL-IJCNLP*.

André F. T. Martins, Noah A. Smith, Eric P. Xing, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of EMNLP*.

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 ACL*.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*.

Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. 1999. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of UAI*.

Jason Naradowsky, Tim Vieira, and David A. Smith. 2012. Grammarless parsing for joint inference. In *Proceedings of COLING*.

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 CoNLL Shared Task Session of EMNLP-CoNLL 2007*.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proceedings of LREC*.

Sebastian Riedel and David A. Smith. 2010. Relaxed marginal inference and its application to dependency parsing. In *Proceedings of NAACL-HLT*.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press.

Alexander M. Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of NAACL-HLT*.

David A. Smith and Jason Eisner. 2006. Minimum-risk annealing for training log-linear models. In *Proceedings of COLING-ACL*.

David A. Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of EMNLP*.

Veselin Stoyanov and Jason Eisner. 2012. Minimum-risk training of approximate CRF-Based NLP systems. In *Proceedings of NAACL-HLT*.

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 AISTATS*.

Martin J. Wainwright. 2006. Estimating the "wrong" graphical model: Benefits in the computation-limited setting. *The Journal of Machine Learning Research*, 7.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of ICML*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, volume 3.