Training dependency parsers by jointly optimizing multiple objectives

Keith Hall **Ryan McDonald** **Jason Katz-Brown** **Michael Ringgaard**
Google Research
{kbhall|ryanmcd|jasonkb|ringgaard}@google.com

Abstract

We present an online learning algorithm for training parsers which allows for the inclusion of multiple objective functions. The primary example is the extension of a standard supervised parsing objective function with additional loss-functions, either based on intrinsic parsing quality or task-specific extrinsic measures of quality. Our empirical results show how this approach performs for two dependency parsing algorithms (graph-based and transition-based parsing) and how it achieves increased performance on multiple target tasks including reordering for machine translation and parser adaptation.

1 Introduction

The accuracy and speed of state-of-the-art dependency parsers has motivated a resumed interest in utilizing the output of parsing as an input to many downstream natural language processing tasks. This includes work on question answering (Wang et al., 2007), sentiment analysis (Nakagawa et al., 2010), MT reordering (Xu et al., 2009), and many other tasks. In most cases, the accuracy of parsers degrades when run on out-of-domain data (Gildea, 2001; McClosky et al., 2006; Blitzer et al., 2006; Petrov et al., 2010). But these accuracies are measured with respect to gold-standard out-of-domain parse trees. There are few tasks that actually depend on the complete parse tree. Furthermore, when evaluated on a downstream task, often the optimal parse output has a model score lower than the best parse as predicted by the parsing model. While this means that we are not properly modeling the downstream task in the parsers, it also means that there is some information from small task or domain-specific data sets which could help direct our search for optimal parameters during parser training. The goal being not necessarily to obtain better parse performance, but to exploit the structure induced from human labeled treebank data while targeting specific extrinsic metrics of quality, which can include task specific metrics or external weak constraints on the parse structure.

One obvious approach to this problem is to employ parser reranking (Collins, 2000). In such a setting, an auxiliary reranker is added in a pipeline following the parser. The standard setting involves training the base parser and applying it to a development set (this is often done in a cross-validated jack-knife training framework). The reranker can then be trained to optimize for the downstream or extrinsic objective. While this will bias the reranker towards the target task, it is limited by the oracle performance of the original base parser.

In this paper, we propose a training algorithm for statistical dependency parsers (Kübler et al., 2009) in which a single model is jointly optimized for a regular supervised training objective over the treebank data as well as a task-specific objective – or more generally an extrinsic objective – on an additional data set. The case where there are both gold-standard trees and a task-specific objective for the entire training set is a specific instance of the larger problem that we address here. Specifically, the algorithm takes the form of an online learner where a training instance is selected and the param-

eters are optimized based on the objective function associated with the instance (either intrinsic or extrinsic), thus jointly optimizing multiple objectives. An update schedule trades-off the relative importance of each objective function. We call our algorithm *augmented-loss training* as it optimizes multiple losses to augment the traditional supervised parser loss.

There have been a number of efforts to exploit weak or external signals of quality to train better prediction models. This includes work on generalized expectation (Mann and McCallum, 2010), posterior regularization (Ganchev et al., 2010) and constraint driven learning (Chang et al., 2007; Chang et al., 2010). The work of Chang et al. (2007) on constraint driven learning is perhaps the closest to our framework and we draw connections to it in Section 5. In these studies the typical goal is to use the weak signal to improve the structured prediction models on the intrinsic evaluation metrics. For our setting this would mean using weak application specific signals to improve dependency parsing. Though we explore such ideas in our experiments, in particular for semi-supervised domain adaptation, we are primarily interested in the case where the weak signal is precisely what we wish to optimize, but also desire the benefit from using both data with annotated parse structures and data specific to the task at hand to guide parser training.

In Section 2 we outline the augmented-loss algorithm and provide a convergence analysis. In Section 3 and 4 we present a set of experiments defining diffent augmented losses covering a task-specific extrinsic loss (MT reordering), a domain adaptation loss, and an alternate intrinsic parser loss. In all cases we show the augmented-loss framework can lead to significant gains in performance. In Section 5 we tie our augmented-loss algorithm to other frameworks for encoding auxiliary information and/or joint objective optimization.

2 Methodology

We present the augmented-loss algorithm in the context of the structured perceptron. The structured perceptron (Algorithm 1) is an on-line learning algorithm which takes as input: 1) a set of training examples $d_i = (x_i, y_i)$ consisting of an input sentence x_i and an output y_i; and 2) a loss-function, $L(\hat{y}, y)$, that measures the cost of predicting output $\hat{y}$ relative to the gold standard y and is usually the 0/1 loss (Collins, 2002). For dependency parser training, this set-up consists of input sentences x and the corresponding gold dependency tree $y \in \mathcal{Y}_x$, where $\mathcal{Y}_x$ is the space of possible parse trees for sentence x. In the perceptron setting, $F_\theta(x) = \arg\max_{y \in \mathcal{Y}_x} \theta \cdot \Phi(y)$ where Φ is mapping from a parse tree y for sentence x to a high dimensional feature space. Learning proceeds by predicting a structured output given the current model, and if that structure is incorrect, updating the model: rewarding features that fire in the gold-standard $\Phi(y_i)$, and discounting features that fire in the predicted output, $\Phi(\hat{y}_i)$.

Algorithm 1 Structured Perceptron

```
{Input data sets: D = {d_1 = (x_1, y_1) ... d_N = (x_N, y_N)}}
{Input 0/1 loss: L(F_θ(x), y) = [F_θ(x) ≠ y ? 1 : 0]}
{Let: F_θ(x) = arg max_{y∈Y} θ · Φ(y)}
{Initialize model parameters: θ = 0⃗}
repeat
  for i = 1 ... N do
    {Compute structured loss}
    ŷ_i = F_θ(x_i)
    if L(ŷ_i, y_i) > 0 then
      {Update model Parameters}
      θ = θ + Φ(y_i) − Φ(ŷ_i)
    end if
  end for
until converged
{Return model θ}
```

The structured perceptron, as given in Algorithm 1, only updates when there is a positive loss, meaning that there was a prediction mistake. For the moment we will abstract away from details such as the precise definition of $F(x)$ and $\Phi(y)$. We will show in the next section that our augmented-loss method is general and can be applied to any dependency parsing framework that can be trained by the perceptron algorithm, such as transition-based parsers (Nivre, 2008; Zhang and Clark, 2008) and graph-based parsers (McDonald et al., 2005).

2.1 Augmented-Loss Training

The augmented-loss training algorithm that we propose is based on the structured perceptron; however, the augmented-loss training framework is a general

mechanism to incorporate multiple loss functions in online learner training. Algorithm 2 is the pseudo-code for the augmented-loss structured perceptron algorithm. The algorithm is an extension to Algorithm 1 where there are 1) multiple loss functions being evaluated $L^1, \ldots, L^M$; 2) there are multiple datasets associated with each of these loss functions $D^1, \ldots, \mathcal{D}^M$; and 3) there is a schedule for processing examples from each of these datasets, where Sched(j, i) is true if the j^{th} loss function should be updated on the i^{th} iteration of training. Note that for data point $d_i^j = (x, y)$, which is the i^{th} training instance of the j^{th} data set, that y does not necessarily have to be a dependency tree. It can either be a task-specific output of interest, a partial tree, or even null, in the case where learning will be guided strictly by the loss L^j. The training algorithm is effectively the same as the perceptron, the primary difference is that if L^j is an extrinsic loss, we cannot compute the standard updates since we do not necessarily know the correct parse (the line indicated by †). Section 2.2 shows one method for updating the parser parameters for extrinsic losses.

In the experiments in this paper, we only consider the case where there are two loss functions: a supervised dependency parsing labeled-attachment loss; and an additional loss, examples of which are presented in Section 3.

Algorithm 2 Augmented-Loss Perceptron

```
{Input data sets}:
  D^1 = {d_1^1 = (x_1^1, y_1^1) ... d_{N^1}^1 = (x_{N^1}^1, y_{N^1}^1)},
  ...
  D^M = {d_1^M = (x_1^M, y_1^M) ... d_{N^M}^M = (x_{N^M}^M, y_{N^M}^M)}
{Input loss functions: L^1 ... L^M}
{Initialize indexes: c_1 ... c_M = 0}
{Initialize model parameters: θ = 0}
i = 0
repeat
  for j = 1 ... M do
    {Check whether to update L^j on iteration i}
    if Sched(j, i) then
      {Compute index of instance – reset if c_j ≡ N^j}
      c_j = [(c_j ≡ N^j) ? 0 : c_j + 1]
      {Compute structured loss for instance}
      if L^j is intrinsic loss then
        ŷ = F_θ(x_{c_j}^j)
        if L^j(ŷ, y_{c_j}^j) > 0 then
          θ = θ + Φ(y_{c_j}^j) − Φ(ŷ)      {y_{c_j}^j is a tree}
        end if
      else if L^j is an extrinsic loss then
        {See Section 2.2}†
      end if
    end if
  end for
  i = i + 1
until converged
{Return model θ}
```

2.2 Inline Ranker Training

In order to make Algorithm 2 more concrete, we need a way of defining the loss and resulting parameter updates for the case when L^j is not a standard supervised parsing loss († from Algorithm 2). Assume that we have a cost function $C(x_i, \hat{y}, y_i)$ which, given a training example (x_i, y_i) will give a score for a parse $\hat{y} \in \mathcal{Y}_{x_i}$ relative to some output y_i. While we can compute the score for any parse, we are unable to determine the features associated with the optimal parse, as y_i need not be a parse tree. For example, consider a machine translation reordering system which uses the parse $\hat{y}$ to reorder the words of x_i, the optimal reordering being y_i. Then $C(x_i, \hat{y}, y_i)$ is a reordering cost which is large if the predicted parse induces a poor reordering of x_i.

We propose a general purpose loss function which is based on parser k-best lists. The inline reranker uses the currently trained parser model θ to parse the external input, producing a k-best set of parses: $F_\theta^{\text{k-best}}(x_i) = \{\hat{y}_1, \ldots, \hat{y}_k\}$. We can compute the cost function $C(x_i, \hat{y}, y_i)$ for all $\hat{y} \in F_\theta^{\text{k-best}}(x_i)$. If the 1-best parse, $\hat{y}_1$, has the lowest cost, then there is no lower cost parse in this k-best list. Otherwise, the lowest-cost parse in $F_\theta^{\text{k-best}}(x_i)$ is taken to be the *correct* output structure y_i, and the 1-best parse is taken to be an incorrect prediction. We can achieve this by substituting the following into Algorithm 2 at line †.

Algorithm 3 Reranker Loss

```
{ŷ_1, ..., ŷ_k} = F_θ^{k-best}(x_i)
τ = min_τ C(x_{c_j}^j, ŷ_τ, y_{c_j}^j)    {τ is min const index}
L^j(ŷ_1, y_{c_j}^j) = C(x_{c_j}^j, ŷ_1, y_{c_j}^j) − C(x_{c_j}^j, ŷ_τ, y_{c_j}^j)
if L^j(ŷ_1, y_{c_j}^j) > 0 then
  θ = θ + Φ(ŷ_τ) − Φ(ŷ_1)
end if
```

Again the algorithm only updates when there is an error – when the 1-best output has a higher cost than any other output in the k-best list – resulting

in positive L^j. The intuition behind this method is that in the presence of only a cost function and a k-best list, the parameters will be updated towards the parse structure that has the lowest cost, which over time will move the parameters of the model to a place with low extrinsic loss.

We exploit this formulation of the general-purpose augmented-loss function as it allows one to include any extrinsic cost function which is dependent of parses. The scoring function used does not need to be factored, requiring no internal knowledge of the function itself. Furthermore, we can apply this to any parsing algorithm which can generate k-best lists. For each parse, we must retain the features associated with the parse (e.g., for transition-based parsing, the features associated with the transition sequence resulting in the parse).

There are two significant differences from the inline reranker loss function and standard reranker training. First, we are performing this decision per example as each data item is processed (this is done in the inner loop of the Algorithm 2). Second, the feedback function for selecting a parse is based on an external objective function. The second point is actually true for many minimum-error-rate training scenarios, but in those settings the model is updated as a post-processing stage (after the base-model is trained).

2.3 Convergence of Inline Ranker Training

A training set $\mathcal{D}$ is loss-separable with margin $\gamma > 0$ if there exists a vector u with $\|u\| = 1$ such that for all $y', y'' \in \mathcal{Y}_x$ and $(x, y) \in \mathcal{D}$, if $L(y', y) < L(y'', y)$, then $u \cdot \Phi(y') - u \cdot \Phi(y'') \geq \gamma$. Furthermore, let $R \geq ||\Phi(y) - \Phi(y')||$, for all y, y'.

Assumption 1. *Assume training set $\mathcal{D}$ is loss-separable with margin γ.*

Theorem 1. *Given Assumption 1. Let m be the number of mistakes made when training the perceptron (Algorithm 2) with inline ranker loss (Algorithm 3) on $\mathcal{D}$, where a mistake occurs for $(x, y) \in \mathcal{D}$ with parameter vector θ when $\exists \hat{y}_j \in F_\theta^{k\text{-best}}(x)$ where $\hat{y}_j \neq \hat{y}_1$ and $L(\hat{y}_j, y) < L(\hat{y}_1, y)$. If training is run indefinitely, then $m \leq \frac{R^2}{\gamma^2}$.*

Proof. Identical to the standard perceptron proof, e.g., Collins (2002), by inserting in loss-separability for normal separability. □

Like the original perceptron theorem, this implies that the algorithm will converge. However, unlike the original theorem, it does not imply that it will converge to a parameter vector θ such that for all $(x, y) \in \mathcal{D}$, if $\hat{y} = \arg\max_{\hat{y}} \theta \cdot \Phi(\hat{y})$ then $L(\hat{y}, y) = 0$. Even if we assume for every x there exists an output with zero loss, Theorem 1 still makes no guarantees. Consider a training set with one instance (x, y). Now, set $k = 2$ for the k-best output list and let $\hat{y}_1$, $\hat{y}_2$, and $\hat{y}_3$ be the top-3 scoring outputs and let $L(\hat{y}_1, y) = 1$, $L(\hat{y}_2, y) = 2$ and $L(\hat{y}_3, y) = 0$. In this case, no updates will ever be made and $\hat{y}_1$ will remain unchanged even though it doesn't have minimal loss. Consider the following assumption:

Assumption 2. *For any parameter vector θ that exists during training, either 1) for all $(x, y) \in \mathcal{D}$, $L(\hat{y}_1, y) = 0$ (or some optimal minimum loss), or 2) there exists at least one $(x, y) \in \mathcal{D}$ where $\exists \hat{y}_j \in F_\theta^{k\text{-best}}(x)$ such that $L(\hat{y}_j, y) < L(\hat{y}_1, y)$.*

Assumption 2 states that for any θ that exists during training, but before convergence, there is at least one example in the training data where k is large enough to include one output with a lower loss when $\hat{y}_1$ does not have the optimal minimal loss. If $k = \infty$, then this is the standard perceptron as it guarantees the optimal loss output to be in the k-best list. But we are assuming something much weaker here, i.e., not that the k-best list will include the minimal loss output, only a single output with a lower loss than the current best guess. However, it is strong enough to show the following:

Theorem 2. *Given Assumption 1 and Assumption 2. Training the perceptron (Algorithm 2) with inline ranker loss (Algorithm 3) on $\mathcal{D}$ 1) converges in finite time, and 2) produces parameters θ such that for all $(x, y) \in \mathcal{D}$, if $\hat{y} = \arg\max_{\hat{y}} \theta \cdot \Phi(\hat{y})$ then $L(\hat{y}, y) = 0$ (or equivalent minimal loss).*

Proof. It must be the case for all $(x, y) \in \mathcal{D}$ that $L(\hat{y}_1, y) = 0$ (and $\hat{y}_1$ is the argmax) after a finite amount of time. Otherwise, by Assumption 2, there exists some x, such that when it is next processed, there would exist an output in the k-best list that had a lower loss, which will result in an additional mistake. Theorem 1 guarantees that this can not continue indefinitely as the number of mistakes is bounded. □

Thus, the perceptron algorithm will converge to optimal minimal loss under the assumption that k is large enough so that the model can keep improving. Note that this does not mean k must be large enough to include a zero or minimum loss output, just large enough to include a better output than the current best hypothesis. Theorem 2, when coupled with Theorem 1, implies that augmented-loss learning will make at most R^2/γ^2 mistakes at training, but does not guarantee the rate at which these mistakes will be made, only that convergence is finite, providing that the scheduling time (defined by Sched()) between seeing the same instance is always finite, which is always true in our experiments.

This analysis does not assume anything about the loss L. Every instance (x, y) can use a different loss. It is only required that the loss for a specific input-output pair is fixed throughout training. Thus, the above analysis covers the case where some training instances use an extrinsic loss and others an intrinsic parsing loss. This also suggests more efficient training methods when extracting the k-best list is prohibitive. One can parse with $k = 2, 4, 8, 16, \ldots$ until an k is reached that includes a lower loss parse. It may be the case that for most instances a small k is required, but the algorithm is doing more work unnecessarily if k is large.

3 Experimental Set-up

3.1 Dependency Parsers

The augmented-loss framework we present is general in the sense that it can be combined with any loss function and any parser, provided the parser can be parameterized as a linear classifier, trained with the perceptron and is capable of producing a k-best list of trees. For our experiments we focus on two dependency parsers.

- **Transition-based**: An implementation of the transition-based dependency parsing framework (Nivre, 2008) using an arc-eager transition strategy and are trained using the perceptron algorithm as in Zhang and Clark (2008) with a beam size of 8. Beams with varying sizes can be used to produce k-best lists. The features used by all models are: the part-of-speech tags of the first four words on the buffer and of the top two words on the stack; the word identities of the first two words on the buffer and of the top word on the stack; the word identity of the syntactic head of the top word on the stack (if available); dependency arc label identities for the top word on the stack, the left and rightmost modifier of the top word on the stack, and the left most modifier of the first word in the buffer (if available). All feature conjunctions are included.

- **Graph-based**: An implementation of graph-based parsing algorithms with an arc-factored parameterization (McDonald et al., 2005). We use the non-projective k-best MST algorithm to generate k-best lists (Hall, 2007), where $k = 8$ for the experiments in this paper. The graph-based parser features used in the experiments in this paper are defined over a word, w_i at position i; the head of this word $w_{\rho(i)}$ where $\rho(i)$ provides the index of the head word; and part-of-speech tags of these words t_i. We use the following set of features similar to McDonald et al. (2005):

isolated features:	$w_i, t_i, w_{\rho(i)}, t_{\rho(i)}$
word-tag pairs:	$(w_i, t_i); (w_{\rho(i)}, t_{\rho(i)})$
word-head pairs:	$(w_i, w_{\rho(i)}), (t_i, t_{\rho(i)})$
word-head-tag triples:	$(t_{\rho(i)}, w_i, t_i)$
	$(w_{\rho(i)}, w_i, t_i)$
	$(w_{\rho(i)}, t_{\rho(i)}, t_i)$
	$(w_{\rho(i)}, t_{\rho(i)}, w_i)$
tag-neighbourhood:	$(t_{\rho(i)}, t_{\rho(i)+1}, t_{i-1}, t_i)$
	$(t_{\rho(i)}, t_{\rho(i)+1}, t_{i+1}, t_i)$
	$(t_{\rho(i)}, t_{\rho(i)-1}, t_{i-1}, t_i)$
	$(t_{\rho(i)}, t_{\rho(i)-1}, t_{i+1}, t_i)$
between features:	$\forall_j\, i < j < \rho(i) \,\|\|\, \rho(i) < j < i$
	$(t_{\rho(i)}, t_j, t_i)$
arc-direction/length :	$(i - \rho(i) > 0, \|i - \rho(i)\|)$

3.2 Data and Tasks

In the next section, we present a set of scoring functions that can be used in the inline reranker loss framework, resulting in a new augmented-loss for each one. Augmented-loss learning is then applied to target a downstream task using the loss functions to measure gains. We show empirical results for two extrinsic loss-functions (optimizing for the downstream task): machine translation and domain adaptation; and for one intrinsic loss-function: an arc-length parsing score. For some experiments we also

measure the standard intrinsic parser metrics unlabeled attachment score (UAS) and labeled attachment score (LAS) (Buchholz and Marsi, 2006).

In terms of treebank data, the primary training corpus is the Penn Wall Street Journal Treebank (PTB) (Marcus et al., 1993). We also make use of the Brown corpus, and the Question Treebank (QTB) (Judge et al., 2006). For PTB and Brown we use standard training/development/testing splits of the data. For the QTB we split the data into three sections: 2000 training, 1000 development, and 1000 test. All treebanks are converted to dependency format using the Stanford converter v1.6 (de Marneffe et al., 2006).

4 Experiments

4.1 Machine Translation Reordering Score

As alluded to in Section 2.2, we use a reordering-based loss function to improve word order in a machine translation system. In particular, we use a system of source-side reordering rules which, given a parse of the source sentence, will reorder the sentence into a target-side order (Collins et al., 2005). In our experiments we work with a set of English-Japanese reordering rules[1] and gold reorderings based on human generated correct reordering of an aligned target sentences. We use a reordering score based on the reordering penalty from the METEOR scoring metric. Though we could have used a further downstream measure like BLEU, METEOR has also been shown to directly correlate with translation quality (Banerjee and Lavie, 2005) and is simpler to measure.

$$\text{reorder-score} = 1 - \frac{\#\text{ chunks} - 1}{\#\text{ unigrams\_matched} - 1}$$

$$\text{reorder-cost} = 1 - \text{reorder-score}$$

All reordering augmented-loss experiments are run with the same treebank data as the baseline (the training portions of PTB, Brown, and QTB). The extrinsic reordering training data consists of 10930 examples of English sentences and their correct Japanese word-order. We evaluate our results on an evaluation set of 6338 examples of similarly created reordering data. The reordering cost, evaluation criteria and data used in our experiments are based on the work of Talbot et al. (2011).

[1] Our rules are similar to those from Xu et al. (2009).

	Exact	Reorder
trans–PTB + Brown + QTB	35.29	76.49
trans–0.5×aug.-loss	38.71	78.19
trans–1.0×aug.-loss	39.02	78.39
trans–2.0×aug.-loss	39.58	78.67
graph–PTB + Brown + QTB	25.71	69.84
graph–0.5× aug.-loss	28.99	72.23
graph–1.0×aug.-loss	29.99	72.88
graph–2.0×aug.-loss	30.03	73.15

Table 1: Reordering scores for parser-based reordering (English-to-Japanese). Exact is the number of correctly reordered sentences. All models use the same treebank-data (PTB, QTB, and the Brown corpus). Results for three augmented-loss schedules are shown: 0.5 where for every two treebank updates we make one augmented-loss update, 1 is a 1-to-1 mix, and 2 is where we make twice as many augmented-loss updates as treebank updates.

Table 1 shows the results of using the reordering cost as an augmented-loss to the standard treebank objective function. Results are presented as measured by the reordering score as well as a coarse exact-match score (the number of sentences which would have correct word-order given the parse and the fixed reordering rules). We see continued improvements as we adjust the schedule to process the extrinsic loss more frequently, the best result being when we make two augmented-loss updates for every one treebank-based loss update.

4.2 Semi-supervised domain adaptation

Another application of the augmented-loss framework is to improve parser domain portability in the presence of partially labeled data. Consider, for example, the case of questions. Petrov et al. (2010) observed that dependency parsers tend to do quite poorly when parsing questions due to their limited exposure to them in the news corpora from the PennTreebank. Table 2 shows the accuracy of two parsers (LAS, UAS and the F1 of the root dependency attachment) on the QuestionBank test data. The first is a parser trained on the standard training sections of the PennTreebank (PTB) and the second is a parser trained on the training portion of the QuestionBank (QTB). Results for both

	LAS	UAS	Root-F1
trans–PTB	67.97	73.52	47.60
trans–QTB	84.59	89.59	91.06
trans–aug.-loss	76.27	86.42	83.41
graph–PTB	65.27	72.72	43.10
graph–QTB	82.73	87.44	91.58
graph–aug.-loss	72.82	80.68	86.26

Table 2: Domain adaptation results. Table shows (for both transition and graph-based parsers) the labeled accuracy score (LAS), unlabeled accuracy score (UAS) and Root-F1 for parsers trained on the PTB and QTB and tested on the QTB. The augmented-loss parsers are trained on the PTB but with a partial tree loss on QTB that considers only root dependencies.

transition-based parsers and graph-based parsers are given. Clearly there is significant drop in accuracy for a parser trained on the PTB. For example, the transition-based PTB parser achieves a LAS of 67.97% relative to 84.59% for the parser trained on the QTB.

We consider the situation where it is possible to ask annotators a single question about the target domain that is relatively easy to answer. The question should be posed so that the resulting answer produces a partially labeled dependency tree. Root-F1 scores from Table 2 suggest that one simple question is "what is the main verb of this sentence?" for sentences that are questions. In most cases this task is straight-forward and will result in a single dependency, that from the root to the main verb of the sentence. We feel this is a realistic partial labeled training setting where it would be possible to quickly collect a significant amount of data.

To test whether such weak information can significantly improve the parsing of questions, we trained an augmented-loss parser using the training set of the QTB stripped of all dependencies except the dependency from the root to the main verb of the sentence. In other words, for each sentence, the parser may only observe a single dependency at training from the QTB – the dependency to the main verb. Our augmented-loss function in this case is a simple binary function: 0 if a parse has the correct root dependency and 1 if it does not. Thus, the algorithm will select the first parse in the k-best list that has the correct root as the proxy to a gold standard parse.[2]

The last row in each section of Table 2 shows the results for this augmented-loss system when weighting both losses equally during training. By simply having the main verb annotated in each sentence – the sentences from the training portion of the QTB – the parser can eliminate half of the errors of the original parser. This is reflected by both the Root-F1 as well as LAS/UAS. It is important to point out that these improvements are not limited to simply better root predictions. Due to the fact that parsing algorithms make many parsing decisions jointly at test time, all such decisions influence each other and improvements are seen across the board. For example, the transition-based PTB parser has an F1 score of 41.22% for verb subjects (nsubj), whereas the augmented-loss parser has an F1 of 73.52%. Clearly improving just a single (and simple to annotate) dependency leads to general parser improvements.

4.3 Average Arc Length Score

The augmented-loss framework can be used to incorporate multiple treebank-based loss functions as well. Labeled attachment score is used as our base model loss function. In this set of experiments we consider adding an additional loss function which weights the lengths of correct and incorrect arcs, the average (labeled) arc-length score:

$$\text{ALS} = \frac{\sum_i \delta(\hat{\rho}_i, \rho_i)(i - \rho_i)}{\sum_i (i - \rho_i)}$$

For each word of the sentence we compute the distance between the word's position i and the position of the words head ρ_i. The arc-length score is the summed length of all those with correct head assignments ($\delta(\hat{\rho}_i, \rho_i)$ is 1 if the predicted head and the correct head match, 0 otherwise). The score is normalized by the summed arc lengths for the sentence. The labeled version of this score requires that the labels of the arc are also correct. Optimizing for dependency arc length is particularly important as parsers tend to do worse on longer dependencies (McDonald and Nivre, 2007) and these dependencies are typically the most meaningful for downstream tasks, e.g., main verb dependencies for tasks

[2] For the graph-based parser one can also find the higest scoring tree with correct root by setting the score of all competing arcs to $-\infty$.

	LAS	UAS	ALS
trans–PTB	88.64	91.64	82.96
trans–unlabeled aug.-loss	88.74	91.91	83.65
trans–labeled aug.-loss	88.84	91.91	83.46
graph–PTB	85.75	88.70	73.88
graph–unlabeled aug.-loss	85.80	88.81	74.26
graph–labeled aug.-loss	85.85	88.93	74.40

Table 3: Results for both parsers on the development set of the PTB. When training with ALS (labeled and unlabeled), we see an improvement in UAS, LAS, and ALS. Furthermore, if we use a labeled-ALS as the metric for augmented-loss training, we also see a considerable increase in LAS.

like information extraction (Yates and Etzioni, 2009) and textual entailment (Berant et al., 2010).

In Table 3 we show results for parsing with the ALS augmented-loss objective. For each parser, we consider two different ALS objective functions; one based on unlabeled-ALS and the other on labeled-ALS. The arc-length score penalizes incorrect long-distance dependencies more than local dependencies; long-distance dependencies are often more destructive in preserving sentence meaning and can be more difficult to predict correctly due to the larger context on which they depend. Combining this with the standard attachment scores biases training to focus on the difficult head dependencies.

For both experiments we see that by adding the ALS augmented-loss we achieve an improvement in LAS and UAS in addition to ALS. The augmented-loss not only helps us improve on the longer dependencies (as reflected in the increased ALS), but also in the main parser objective function of LAS and UAS. Using the labeled loss function provides better reinforcement as can be seen in the improvements over the unlabeled loss-function. As with all experiments in this paper, the graph-based parser baselines are much lower than the transition-based parser due to the use of arc-factored features. In these experiments we used an inline-ranker loss with 8 parses. We experimented with larger sizes (16 and 64) and found very similar improvements: for example, the transition parser's LAS for the labeled loss is 88.68 and 88.84, respectively).

We note that ALS can be decomposed locally and could be used as the primary objective function for parsing. A parse with perfect scores under ALS and LAS will match the gold-standard training tree. However, if we were to order incorrect parses of a sentence, ALS and LAS will suggest different orderings. Our results show that by optimizing for losses based on a combination of these metrics we train a more robust parsing model.

5 Related Work

A recent study by Katz-Brown et al. (2011) also investigates the task of training parsers to improve MT reordering. In that work, a parser is used to first parse a set of manually reordered sentences to produce k-best lists. The parse with the best reordering score is then fixed and added back to the training set and a new parser is trained on resulting data. The method is called *targeted self-training* as it is similar in vein to self-training (McClosky et al., 2006), with the exception that the new parse data is targeted to produce accurate word reorderings. Our method differs as it does not statically fix a new parse, but dynamically updates the parameters and parse selection by incorporating the additional loss in the inner loop of online learning. This allows us to give guarantees of convergence. Furthermore, we also evaluate the method on alternate extrinsic loss functions.

Liang et al. (2006) presented a perceptron-based algorithm for learning the phrase-translation parameters in a statistical machine translation system. Similar to the inline-ranker loss function presented here, they use a k-best lists of hypotheses in order to identify parameters which can improve a global objective function: BLEU score. In their work, they are interested in learning a parameterization over translation phrases (including the underlying word-alignment) which optimizes the BLEU score. Their goal is considerably different; they want to incorporate additional features into their model and define an objective function which allows them to do so; whereas, we are interested in allowing for multiple objective functions in order to adapt the parser model parameters to downstream tasks or alternative intrinsic (parsing) objectives.

The work that is most similar to ours is that of Chang et al. (2007), who introduced the Constraint Driven Learning algorithm (CODL). Their algorithm specifically optimizes a loss function with

the addition of constraints based on unlabeled data (what we call extrinsic datasets). For each unlabeled example, they use the current model along with their set of constraints to select a set of k automatically labeled examples which best meet the constraints. These induced examples are then added to their training set and, after processing each unlabeled dataset, they perform full model optimization with the concatenation of training data and newly generated training items. The augmented-loss algorithm can be viewed as an online version of this algorithm which performs model updates based on the augmented-loss functions directly (rather than adding a set of examples to the training set). Unlike the CODL approach, we do not perform complete optimization on each iteration over the unlabeled dataset; rather, we incorporate the updates in our online learning algorithm. As mentioned earlier, CODL is one example of learning algorithms that use weak supervision, others include Mann and McCallum (2010) and Ganchev et al. (2010). Again, these works are typically interested in using the extrinsic metric – or, in general, extrinsic information – to optimize the intrinsic metric in the absence of any labeled intrinsic data. Our goal is to optimize both simultaneously.

The idea of jointly training parsers to optimize multiple objectives is related to joint learning and inference for tasks like information extraction (Finkel and Manning, 2009) and machine translation (Burkett et al., 2010). In such works, a large search space that covers both the space of parse structures and the space of task-specific structures is defined and parameterized so that standard learning and inference algorithms can be applied. What sets our work apart is that there is still just a single parameter set that is being optimized – the parser parameters. Our method only uses feedback from task specific objectives in order to update the parser parameters, guiding it towards better downstream performance. This is advantageous for two reasons. First, it decouples the tasks, making inference and learning more efficient. Second, it does not force arbitrary paraemter factorizations in order to define a joint search space that can be searched efficiently.

Finally, augmented-loss training can be viewed as multi-task learning (Caruana, 1997) as the model optimizes multiple objectives over multiple data sets with a shared underlying parameter space.

6 Discussion

The empirical results show that incorporating an augmented-loss using the inline-ranker loss framework achieves better performance under metrics associated with the external loss function. For the intrinsic loss, we see that the augmented-loss framework can also result in an improvement in parsing performance; however, in the case of ALS, this is due to the fact that the loss function is very closely related to the standard evaluation metrics of UAS and LAS.

Although our analysis suggests that this algorithm is guaranteed to converge only for the separable case, it makes a further assumption that if there is a better parse under the augmented-loss, then there must be a lower cost parse in the k-best list. The empirical evaluation presented here is based on a very conservative approximation by choosing lists with at most 8 parses. However, in our experiments, we found that increasing the size of the lists did not significantly increase our accuracy under the external metrics. If we do have at least one improvement in our k-best lists, the analysis suggests that this is enough to move in the correct direction for updating the model. The assumption that there will always be an improvement in the k-best list if there is some better parse breaks down as training continues. We suspect that an increasing k, as suggested in Section 2.3, will allow for continued improvements.

Dependency parsing, as presented in this paper, is performed over (k-best) part-of-speech tags and is therefore dependent on the quality of the tagger. The experiments presented in this paper made use of a tagger trained on the source treebank data which severely limits the variation in parses. The augmented-loss perceptron algorithm presented here can be applied to any online learning problem, including part-of-speech tagger training. To build a dependency parser which is better adapted to a downstream task, one would want to perform augmented-loss training on the tagger as well.

7 Conclusion

We introduced the augmented-loss training algorithm and show that the algorithm can incorporate

additional loss functions to adapt the model towards extrinsic evaluation metrics. Analytical results are presented that show that the algorithm can optimize multiple objective functions simultaneously. We present an empirical analysis for training dependency parsers for multiple parsing algorithms and multiple loss functions.

The augmented-loss framework supports both intrinsic and extrinsic losses, allowing for both combinations of objectives as well as multiple sources of data for which the results of a parser can be evaluated. This flexibility makes it possible to tune a model for a downstream task. The only requirement is a metric which can be defined over parses of the downstream data. Our dependency parsing results show that we are not limited to increasing parser performance via more data or external domain adaptation techniques, but that we can incorporate the downstream task into parser training.

Acknowledgements: We would like to thank Kuzman Ganchev for feedback on an earlier draft of this paper as well as Slav Petrov for frequent discussions on this topic.

References

S. Banerjee and A. Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*.

J. Berant, I. Dagan, and J. Goldberger. 2010. Global learning of focused entailment graphs. In *Proc. of ACL*.

J. Blitzer, R. McDonald, and F. Pereira. 2006. Domain adaptation with structural correspondence learning. In *Proc. of EMNLP*.

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of CoNLL*.

D. Burkett, J. Blitzer, and D. Klein. 2010. Joint parsing and alignment with weakly synchronized grammars. In *Proc. of NAACL*.

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

M.W. Chang, L. Ratinov, and D. Roth. 2007. Guiding semi-supervision with constraint-driven learning. In *Proc. of ACL*.

M. Chang, D. Goldwasser, D. Roth, and V. Srikumar. 2010. Structured output learning with indirect supervision. In *Proc. of ICML*.

M. Collins, P. Koehn, and I. Kučerová. 2005. Clause restructuring for statistical machine translation. In *Proc. of ACL*.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proc. of ICML*.

M. Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of ACL*.

M.C. de Marneffe, B. MacCartney, and C. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proc. of LREC*, Genoa, Italy.

J.R. Finkel and C.D. Manning. 2009. Joint parsing and named entity recognition. In *Proc. of NAACL*.

K. Ganchev, J. Graça, J. Gillenwater, and B. Taskar. 2010. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*.

D. Gildea. 2001. Corpus variation and parser performance. In *Proc. of EMNLP*.

K. Hall. 2007. k-best spanning tree parsing. In *Proc. of ACL*, June.

J. Judge, A. Cahill, and J. Van Genabith. 2006. Questionbank: Creating a corpus of parse-annotated questions. In *Proc. of ACL*, pages 497–504.

J. Katz-Brown, S. Petrov, R. McDonald, D. Talbot, F. Och, H. Ichikawa, M. Seno, and H. Kazawa. 2011. Training a parser for machine translation reordering. In *Proc. of EMNLP*.

S. Kübler, R. McDonald, and J. Nivre. 2009. *Dependency parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

P. Liang, A. Bouchard-Ct, D. Klein, and B. Taskar. 2006. An end-to-end discriminative approach to machine translation. In *Proc. of COLING/ACL*.

G.S. Mann and A. McCallum. 2010. Generalized Expectation Criteria for Semi-Supervised Learning with Weakly Labeled Data. *The Journal of Machine Learning Research*, 11:955–984.

M. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19:313–330.

D. McClosky, E. Charniak, and M. Johnson. 2006. Reranking and self-training for parser adaptation. In *Proc. of ACL*.

R. McDonald and J. Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proc. of EMNLP-CoNLL*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proc. of ACL*.

T. Nakagawa, K. Inui, and S. Kurohashi. 2010. Dependency tree-based sentiment classification using crfs with hidden variables. In *Proc. of NAACL*.

J. Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

S. Petrov, P.C. Chang, M. Ringgaard, and H. Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proc. of EMNLP*, pages 705–713.

D. Talbot, H. Kazawa, H. Ichikawa, J. Katz-Brown, M. Seno, and F. Och. 2011. A lightweight evaluation framework for machine translation reordering. In *Proc. of the Sixth Workshop on Statistical Machine Translation*.

M. Wang, N.A. Smith, and T. Mitamura. 2007. What is the Jeopardy model? A quasi-synchronous grammar for QA. In *Proc. of EMNLP-CoNLL*.

P. Xu, J. Kang, M. Ringgaard, and F. Och. 2009. Using a dependency parser to improve SMT for Subject-Object-Verb languages. In *Proc. of NAACL*.

A. Yates and O. Etzioni. 2009. Unsupervised methods for determining object and relation synonyms on the web. *Journal of Artificial Intelligence Research*, 34(1):255–296.

Y. Zhang and S. Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In *Proc. of EMNLP*, pages 562–571.