How to Train Dependency Parsers with Inexact Search for Joint Sentence Boundary Detection and Parsing of Entire Documents

Anders Björkelund and **Agnieszka Faleńska** and **Wolfgang Seeker** and **Jonas Kuhn**
Institut für Maschinelle Sprachverarbeitung
University of Stuttgart
{anders,falensaa,seeker,jonas}@ims.uni-stuttgart.de

Abstract

We cast sentence boundary detection and syntactic parsing as a joint problem, so an entire text document forms a training instance for transition-based dependency parsing. When trained with an early update or max-violation strategy for inexact search, we observe that only a tiny part of these very long training instances is ever exploited. We demonstrate this effect by extending the ArcStandard transition system with swap for the joint prediction task. When we use an alternative update strategy, our models are considerably better on both tasks and train in substantially less time compared to models trained with early update/max-violation. A comparison between a standard pipeline and our joint model furthermore empirically shows the usefulness of syntactic information on the task of sentence boundary detection.

1 Introduction

Although punctuation mostly provides reliable cues for segmenting longer texts into sentence units, human readers are able to exploit their understanding of the syntactic and semantic structure to (re-)segment input in the absence of such cues.

When working with carefully copy-edited text documents, sentence boundary detection can be viewed as a minor preprocessing task in Natural Language Processing, solvable with very high accuracy. However, when dealing with the output of automatic speech recognition or "noisier" texts such as blogs and emails, non-trivial sentence segmentation issues do occur. Dridan and Oepen (2013), for example, show how much impact fully automatic preprocessing can have on parsing quality for well-edited and less-edited text.

Two possible strategies to approach this problem are (i) to exploit other cues for sentence boundaries, such as prosodic phrasing and intonation in speech (e.g., Kolář et al. (2006)) or formatting cues in text documents (Read et al., 2012), and (ii) to emulate the human ability to exploit syntactic competence for segmentation. We focus here on the latter, which has received little attention, and propose to cast sentence boundary detection and syntactic (dependency) parsing as a joint problem, such that segmentations that would give rise to suboptimal syntactic structures can be discarded early on.

A joint model for parsing and sentence boundary detection by definition operates on documents rather than single sentences, as is the standard case for parsing. The task is illustrated in Figure 1, which shows the beginning of a document in the Switchboard corpus, a collection of transcribed telephone dialogues. The parser must predict the syntactic structure of the three sentences as well as the start points of each sentence.[1]

The simple fact that documents are considerably longer than sentences, often by orders of magnitude, creates some interesting challenges for a joint system. First of all, the decoder needs to handle long inputs efficiently. This problem is easily solved by using transition-based decoders, which excel in this kind of setting due to their incremental approach and their low theoretical complexity. Specifically, we use a transition-based decoder that extends the Swap transition system of Nivre (2009) in order to introduce sentence boundaries during the parsing process. The parser performs inexact search for the optimal structure by

[1]We chose this example for its brevity. For this particular example, the task of sentence boundary prediction could be solved easily with speaker information since the second sentence is from another speaker's turn. The interesting cases involve sentence segmentation within syntactically complex turns of a single speaker.

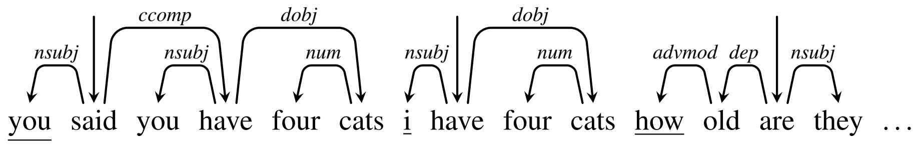

ccomp
dobj
dobj
nsubj
nsubj
num
nsubj
num
advmod
dep
nsubj
you said you have four cats i have four cats how old are they ...

Figure 1: The beginning of a sample document from the Switchboard corpus. Tokens that start a sentence are underlined. The task is to predict syntactic structure and sentence boundaries jointly.

maintaining a beam of several candidate derivations throughout the parsing process.

We will show in this paper that, besides efficient decoding, a second, equally significant challenge lies in the way such a parser is trained. Normally, beam-search transition-based parsers are trained with structured perceptrons using either *early update* (Zhang and Clark, 2008; Collins and Roark, 2004) or *max-violation updates* (Huang et al., 2012). Yet our analysis demonstrates that neither of these update strategies is appropriate for training on very long input sequences as they discard a large portion of the training data.[2] A significant part of the training data is therefore never used to train the model. As a remedy to this problem, we instead use an adaptation of the update strategy in Björkelund and Kuhn (2014). They apply early update in a coreference resolution system and observe that the task is inherently so difficult that the correct item practically never stays in the beam. So early updates are unable to exploit the full instances during training. They propose to apply the updates iteratively on the same document until the full document has been observed. In our case, i.e. when parsing entire documents, the problem is similar in that early updates do not reach the point where the learning algorithm exploits the full training data within reasonable time. Training instead with the iterative update strategy gives us significantly better models in substantially less training time.

The second contribution in this paper is to demonstrate empirically that syntactic information can make up to a large extent for missing or unreliable cues from punctuation. The joint system implements this hypothesis and allows us to test the influence of syntactic information on the prediction of sentence boundaries as compared to a pipeline baseline where both tasks are performed independently of each other. For our analysis, we use the Wall Street Journal as the standard benchmark set and as a representative for copy-edited text. We also use the Switchboard corpus of transcribed dialogues as a representative for data where punctuation cannot give clues to a sentence boundary predictor (other types of data that may show this property to varying degrees are web content data, e.g. forum posts or chat protocols, or (especially historical) manuscripts). While the Switchboard corpus gives us a realistic scenario for a setting with unreliable punctuation, the syntactic complexity of telephone conversations is rather low compared to the Wall Street Journal. Therefore, as a controlled experiment for assessing how far syntactic competence alone can take us if we stop trusting punctuation and capitalization entirely, we perform joint sentence boundary detection/parsing on a lower-cased, no-punctuation version of the Wall Street Journal. In this setting, where the parser must rely on syntactic information alone to predict sentence boundaries, syntactic information makes a difference of 10 percentage point absolute for the sentence boundary detection task, and two points for labeled parsing accuracy.

2 Transition system

We start from the ArcStandard system extended with a swap transition to handle non-projective arcs (Nivre, 2009). We add a transition SB (for sentence boundary) that flags the front of the buffer as the beginning of a new sentence. SB blocks the SHIFT transition until the stack has been reduced and a tree has been constructed, which prevents the system from introducing arcs between separate sentences. To track the predicted sentence boundaries, we augment the configurations with a set S to hold the predicted sentence boundaries. Conceptually this leads to a representation where a document has a single artificial root

[2]We make one simplifying assumption in our experimental setup by assuming gold tokenization. Tokenization is often taken for granted, mostly because it is a fairly easy task in English. For a realistic setting, tokenization would have to be predicted as well, but since we are interested in the effect of long sequences on training, we do not complicate our setting by including tokenization.

Transition				Preconditions
LeftArc	$(\sigma\|s_1\|s_0, \beta, A, S)$	$\Rightarrow$	$(\sigma\|s_0, \beta, A \cup \{s_0 \rightarrow s_1\}, S)$	$s_1 \neq 0$
RightArc	$(\sigma\|s_1\|s_0, \beta, A, S)$	$\Rightarrow$	$(\sigma\|s_1, \beta, A \cup \{s_1 \rightarrow s_0\}, S)$	
Shift	$(\sigma, b_0\|\beta, A, S)$	$\Rightarrow$	$(\sigma\|b_0, \beta, A, S)$	$b_0 \neq \textsc{Last}(S) \vee \|\sigma\| = 1 \vee \textsc{Swapped}(\beta)$
Swap	$(\sigma\|s_1\|s_0, \beta, A, S)$	$\Rightarrow$	$(\sigma\|s_0, s_1\|\beta, A, S)$	$s_1 < s_0$
Sb	$(\sigma, b_0\|\beta, A, S)$	$\Rightarrow$	$(\sigma, b_0\|\beta, A, S \cup \{b_0\})$	$\textsc{Last}(S) < b_0 \wedge \neg\textsc{Swapped}(\beta)$

Figure 2: Transition system. $\sigma|s_1|s_0$ denotes the stack with s_0 and s_1 on top, $b_0|\beta$ denotes the buffer with b_0 in front. $\textsc{Last}(S)$ denotes the most recent sentence boundary, and $\textsc{Swapped}(\beta)$ is true iff the buffer contains swapped items.

node that replaces the artificial root nodes for individual sentences.

The transition types of the system are shown in Figure 2. The configurations consist of four data structures: the stack σ, the input buffer β, the set of constructed arcs A, and the set of sentence boundaries S. LeftArc, RightArc, and Swap have the same semantics and preconditions as in Nivre (2009). We modify the preconditions of Shift in order to block shifts when necessary. Whether shift is allowed can be categorized into three cases subject to the most recently predicted sentence boundary:

- If $\textsc{Last}(S) < b_0$: The last predicted sentence boundary has already been shifted onto the stack. At this point, the system is building a new sentence and has not yet decided where it ends. Shift is therefore allowed.
- If $\textsc{Last}(S) > b_0$: This situation can only occur if the system predicted a sentence boundary and subsequently made a Swap. $\textsc{Last}(S)$ then denotes the end of the current sentence and is deeper in the buffer than b_0. Thus Shift is allowed since b_0 belongs to the current sentence.
- If $\textsc{Last}(S) = b_0$: The system must complete the current sentence by reducing the stack before it can continue shifting and Shift is generally not allowed, with two exceptions. (1) If the stack consists only of the root (i.e., $|\sigma| = 1$), the current sentence has been completed and the system is ready to begin parsing the next one. (2) If b_0 denotes the beginning of a new sentence, but it has been swapped back onto the buffer, then it belongs to the same sentence as the tokens currently on the stack.

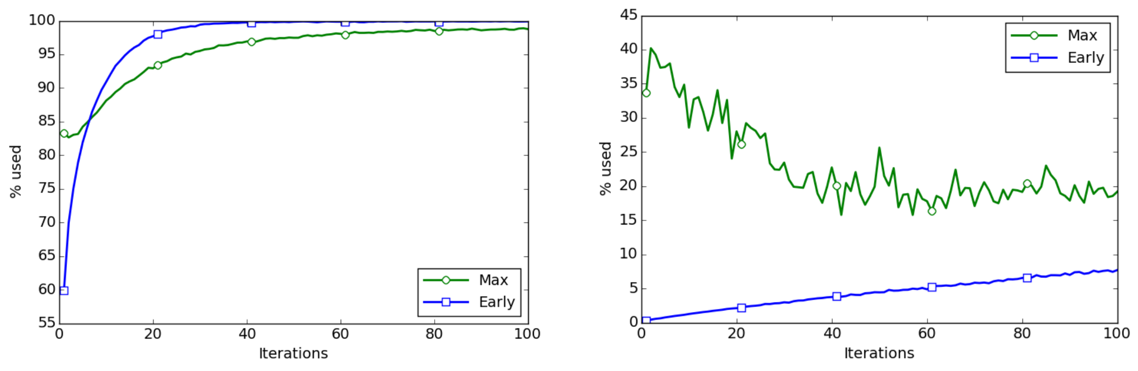
The preconditions for Sb are straightforward. It is only allowed if the current b_0 is ahead of the most recently predicted sentence boundary. Additionally, the transition is not allowed if b_0 has been swapped out from the stack. If it were, then b_0 would be part of the following sentence and sentences would no longer be continuous.

Extending the transition system to also handle sentence boundaries does not affect the computational complexity. While the swap transition system has worst case $O(n^2)$ complexity, Nivre (2009) shows that swaps are rare enough that the system maintains a linear time complexity on average. A naive implementation of the configurations that make the arc set A and the sentence boundary set S explicit could result in configurations that require linear time for copying during beam search. Goldberg et al. (2013) show how this problem can be circumvented in the case of a sentence-based parser. Instead of making the arc set explicit, the arcs are reconstructed after parsing by following back-pointers to previous states. Only a small set of arcs required for feature extraction are saved in the states. We note that the same trick can be applied to avoid keeping an explicit representation of S since the system only needs to know the last predicted sentence boundary.

Snt 1	sh sh la sh sh la sh sh sb$_{\text{early}}$ la ra ra ra sb$_{\text{late}}$
Snt 2	sh sh la sh sh sb$_{\text{early}}$ la ra ra sb$_{\text{late}}$
Snt 3	sh sh la sh la sh sb$_{\text{early}}$ ra ra sb$_{\text{late}}$

Table 1: Transition sequences including sentence boundary transitions for the example in Figure 1.

Oracle. Since each sentence constitutes its own subtree under the root node, a regular sentence-based oracle can be used to derive the oracle transition sequence for complete documents. Specifically, we apply the sentence-based oracle to each sentence, and then join the sequences with a Sb transition in between. In the oracle sequences derived this way we can either apply the Sb transition as *late* as possible, requiring the system to completely reduce the stack before introducing a sentence boundary. Alternatively, sentence boundaries can be introduced as *early* as possible, i.e., applying the Sb transition as soon as b_0 starts a new sentence. Table 1 shows this difference in the



(a) sentences (b) documents

Figure 3: Average length of training sequences used during training for early update and max violation.

oracle transitions of each sentence from Figure 1. During preliminary experiments we compared the two alternatives and found that the early version performed better than the late.[3]

3 Learning

We focus on training structured perceptrons for search-based dependency parsing. Here, the score of a parse is defined as the scalar product of a weight vector w and a global feature vector Φ. The feature vector in turn is defined as the sum of *local* feature vectors ϕ, corresponding to the features extracted for a single transition t given a configuration c. During prediction we thus aim to obtain

$$\begin{aligned}\hat{y} &= \operatorname*{arg\,max}_{y \in \mathcal{Y}} \; \Phi(y) \cdot w \\ &= \operatorname*{arg\,max}_{y \in \mathcal{Y}} \sum_{(t,c) \in \textsc{TrSeq}(y)} \phi(t,c) \cdot w \qquad (1)\end{aligned}$$

where TRSEQ represents the sequence of configurations and transitions executed to obtain the tree y. As the space of possible transition sequences is too large to search exhaustively, we use beam search for approximate search.

Early Update. Using approximate search while training structured perceptrons is complicated by the fact that the correct solution may actually obtain the highest score given the current model, but it was pruned off by the search procedure and therefore never considered. Collins and Roark (2004) solve this by halting search as soon as the correct solution is pruned and then making an *early update* on the partial sequences. Intuitively this makes sense, since once the correct solution is no longer reachable, it makes no sense to continue searching.

Max-violation updates. Huang et al. (2012) note that early updates require a considerable number of training iterations as it often discards a big portion of the training data. Moreover, they show that updates covering a greater subsequence can also be valid and define the *max-violation* update. Specifically, max-violation updates extend the beam beyond early updates and apply updates where the maximum difference in scores as defined by Equation (1) between the correct solution and the best prediction (i.e., the maximal violation) is used for update. They show that this leads to faster convergence compared to early update.

The curse of long sequences. Neither early nor max-violation updates commit to using the full training sequence for updates. In standard sentence-level tasks such as part-of-speech tagging or sentence-based dependency parsing these updates suffice and reasonably quickly reach a level where all or almost all of the training sequences are used for training. An entire document, however, may be composed of tens or hundreds of sentences, leading to transition sequences that are orders of magnitude longer.

To illustrate the difference between sentence-level and document-level parsing Figure 3 shows plots of the average percentage of the gold training sequences that are being used as training progresses on the Switchboard training set.

The left plot shows the parser trained on sen-

[3]One could also imagine leaving the decision of when to apply SB latent and let the machine learning decide. However, preliminary experiments again suggested that this strategy was inferior to the earliest possible point.

tences, the right one when it is trained on documents (where it also has to predict sentence boundaries). On the sentence level we see that both update strategies quite quickly reach close to 100%, i.e., they see more or less complete transition sequences during training. On the document level the picture is considerably different. The average length of seen transition sequences never even goes above 50%. In other words, more than half of the training data is never used. Early update shows a slow increase over time, presumably because the parser sees a bit more of every training instance at every iteration and therefore advances. However, max violation starts off using much more training data than early update, but then drops and settles around 20%. This illustrates the fact that max violation does not commit to exploiting more training data, but rather selects the update which constitutes the maximum violation irrespective of how much of the instance is being used. Empirically, even though the percentage of used training data decreases over iterations, max violation is still profiting from more iterations in the document-level task (cf. Figure 4 in Section 4).

Delayed LaSO. To solve the problem with the discarded training data, we follow Björkelund and Kuhn (2014) and apply the DLASO[4] update. This idea builds on early update, but crucially differs in the sense that the remainder of a training sequence is not discarded when a mistake is made. Rather, the corresponding update is stored and the beam is reseeded with the correct solution. This enables the learning algorithm to exploit the full training data while still making sound updates (or, using the terminology of Huang et al. (2012), a number of updates that are all *violations*).

Pseudocode for DLASO is shown in Algorithm 1. Similar to early update it performs beam search until the correct item falls off the beam (lines 9-12). Here, early update would halt, update the weights w and move on to the next instance. Instead, DLASO computes the corresponding update, i.e., a change in the w, and stores it away. It then resets the beam to the correct solution c_i, and continues beam search. This procedure is repeated until the end of a sequence, with a final check for correctness after search has finished (line 15). After a complete pass-through of the training instance an update is made if any updates were recorded during beam search (lines 17-18).

[4]*Delayed Learning as Search Optimization*

Algorithm 1 DLaSO

```
Input: Training data $D = \{(x_i, y_i)\}_{i=1}^n$, epochs $T$, beam size $B$.
Output: Weight vector $w$.
 1: $w = 0$
 2: for $t \in 1..T$ do
 3:     for $(x, y) \in D$ do
 4:         $c_{0..n} = \textsc{Oracle}(y)$
 5:         Beam $= \{c_0\}$
 6:         Updates $= \{\}$
 7:         for $i \in 1 .. (n-1)$ do
 8:             Beam $= \textsc{ExpandAndFilter}(\text{Beam}, B)$
 9:             if $c_i \notin$ Beam then
10:                 $\hat{y} = \textsc{Best}(\text{Beam})$
11:                 Updates $=$ Updates $\cup\ \textsc{CalcUpdate}(c_i, \hat{y})$
12:                 Beam $= \{c_i\}$
13:         Beam $= \textsc{ExpandAndFilter}(\text{Beam}, B)$
14:         $\hat{y} = \textsc{Best}(\text{Beam})$
15:         if $c_n \neq \hat{y}$ then
16:             Updates $=$ Updates $\cup\ \textsc{CalcUpdate}(c_n, \hat{y})$
17:         if $|\text{Updates}| > 0$ then
18:             $w = \textsc{ApplyUpdates}(w, \text{Updates})$
19: return $w$
```

The DLASO update is closely related to LASO (Daumé III and Marcu, 2005), but differs in that it delays the updates until the full instance has been decoded. Björkelund and Kuhn (2014) show that the difference is important, as it prevents the learning algorithm from getting feedback within instances. Without the delay the learner can bias the weights for rare (e.g., lexicalized) features that occur within a single instance which renders the learning setting quite different from test time inference where no such feedback is available.

4 Experimental setup

Data sets. We experiment with two parts of the English Penn Treebank (Marcus et al., 1993). We use the Wall Street Journal (WSJ) as an example of copy-edited newspaper-quality texts with proper punctuation and capitalized sentences. We also use the Switchboard portion which consists of (transcribed) telephone conversations between strangers. Following previous work on Switchboard we lowercase all text and remove punctuation and disfluency markups.

We use sections 2-21 of the WSJ for training, 24 as development set and 23 as test set. For Switchboard we follow Charniak and Johnson (2001). We convert both data sets to Stanford dependencies with the Stanford dependency converter (de Marneffe et al., 2006). We predict part-of-speech tags with the CRF tagger MARMOT (Müller et al., 2013) and annotate the training sets via 10-fold jackknifing. Depending on the experimental scenario we use MarMoT in two different settings – standard sentence-level where we train and apply it on sentences, and document-level where a whole document is fed to the tagger, implicitly treating it as a single very long sentence.

Sentence boundary detection. We work with two well-established sentence boundary detection baselines. Following (Read et al., 2012) we use the tokenizer from the Stanford CoreNLP (Manning et al., 2014) and the sentence boundary detector from OpenNLP[5] which has been shown to achieve state-of-the-art results on WSJ. We evaluate the performance of sentence boundary detection on the token level using F-measure (F_1).[6]

Typical sentence boundary detectors such as CoreNLP or OpenNLP focus on punctuation marks and are therefore inapplicable to data like Switchboard that does not originally include punctuation. In such cases CRF taggers are commonly selected as baselines, e.g. for punctuation prediction experiments (Zhang et al., 2013a). We therefore introduce a third baseline using MarMoT. For this, we augment the POS tags with information to indicate if a token starts a new sentence or not. We prepare the training data accordingly and train the document-level sequence labeler on them. Table 2 shows the accuracies of all baseline systems on the development sets. For WSJ all three algorithms achieve similar results which shows that MarMoT is a competitive baseline. As can be seen, predicting sentence boundaries for the Switchboard dataset is a more difficult task than for well-formatted text like the WSJ.

	WSJ	Switchboard
OpenNLP	98.09	–
CoreNLP	98.60	–
MarMoT	98.21	71.78

Table 2: Results (F_1) for baselines for sentence boundary detection on dev sets.

Parser implementation. Our parser implements the labeled version of the transition system described in Section 2 with a default beam size of 20. We use the oracle by Nivre et al. (2009) to create transition sequences for each sentence of a document, and then concatenate them with Sb transitions that occur as early as possible (cf. Section 2). The feature set is based on previous work (Zhang and Nivre, 2011; Bohnet and Kuhn, 2012; Bohnet et al., 2013) and was developed for a sentence-based parser for the WSJ. We made initial experiments trying to introduce new features aimed at capturing sentence boundaries such as trying to model verb subcategorization or sentence length, however none of these proved useful compared to the baseline feature set. Following the line of work by Bohnet et al., we use the passive-aggressive algorithm (Crammer et al., 2006) instead of the vanilla perceptron, parameter averaging (Collins, 2002), and a hash function to map features (Bohnet, 2010).[7]

5 Analysis

Comparison of training methods. Figure 4 shows learning curves of the different training algorithms where sentence boundary F_1 and parsing accuracy LAS are plotted as a function of training iterations. The plots show performance for early update, max-violation, and DLaSO updates. In addition, a greedy version of the parser is also included. The greedy parser uses a plain averaged perceptron classifier that is trained on all the training data. The straight dashed line corresponds to the MarMoT baseline.

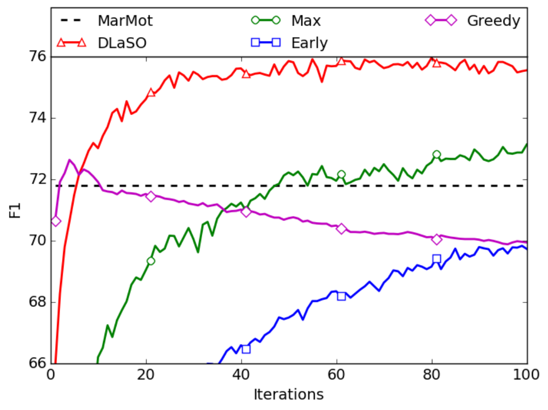
While the greedy parser, DLaSO, and the MarMoT baseline all exploit the full training data during training, early update and max-violation do not (as shown in Section 3). This fact has a direct impact on the performance of these systems. DLaSO reaches a plateau rather quickly, whereas even after 100 iterations, early update and max-violation perform considerably worse.[8] We also see that the greedy parser quickly reaches an optimum and then starts degrading, presumably due to overfitting. It is noteworthy, however, that max-violation needs something between 40 to 60 iterations until it reaches a level similar to the greedy parsers optimal value. This effect is quite different from single sentence parsing scenarios, where it is known that beam search parsers easily outperform the greedy counterparts, requiring not nearly as many training iterations.

[5]`http://opennlp.apache.org`

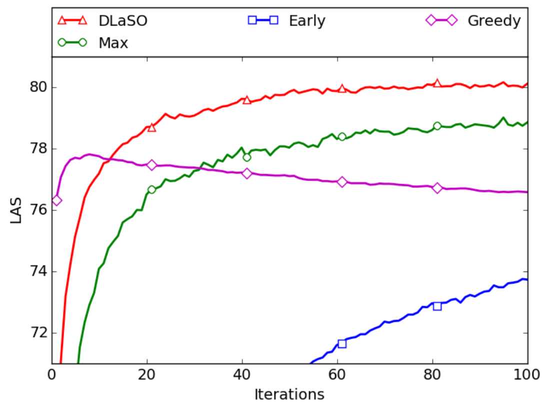
[6]A true positive is defined as a token that was correctly predicted to begin a new sentence.

[7]We make the source code of the parser available on the first author's website.

[8]The x-axis is cut at 100 iterations for simplicity. Although early update and max-violation still are growing at this point, the overall effect does not change – even after 200 iterations the DLaSO update outperforms the other two by at least two points absolute.



(a) sentence boundary detection



(b) parsing

Figure 4: Performance of different update strategies on the Switchboard development set.

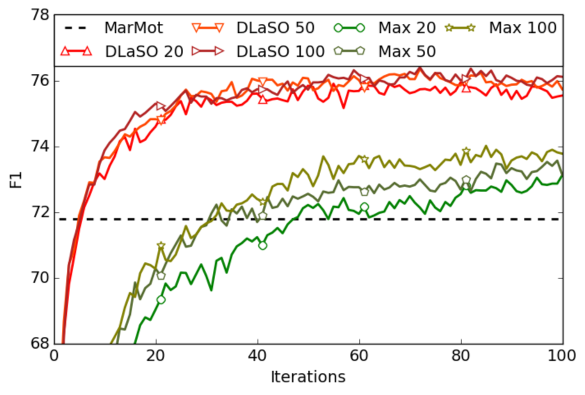


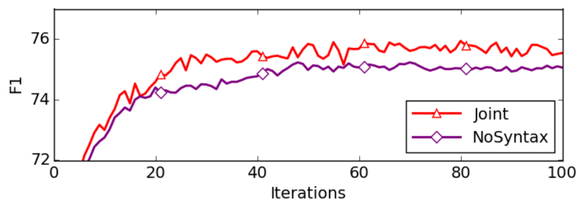
Figure 5: The effect of increasing beam size.

Increasing the beam size. Intuitively, using a bigger beam size might alleviate the problem of discarded training data and enable max-violation to exploit more training data. Figure 5 shows the sentence boundary F_1 as a function of training iterations for different beam sizes for DLASO and max-violation. For DLASO, we see that a bigger beam provides a slight improvement. Max-violation shows a greater correlation between greater beam size and improved F_1. However, even with a beam of size 100 max-violation is nowhere near DLASO. In theory a beam size orders of magnitude greater may rival DLASO but as the beam size directly influences the time complexity of the parser, this is not a viable option.

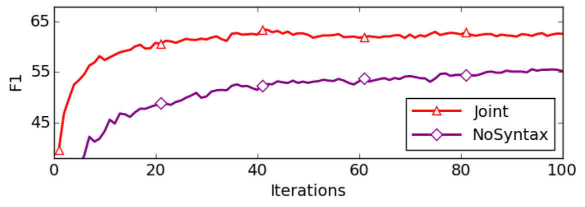
Does syntax help? One of the underlying assumptions of the joint model is our expectation that access to syntactic information should support the model in finding the sentence boundaries. We have already seen that the joint parser outperforms the MARMOT baseline by a big margin in terms of sentence boundary F_1 (Figure 4). However, the comparison is not entirely fair as the two systems use different feature sets and learning algorithms.

To properly measure the effect of syntactic information on the sentence boundary detection task, we therefore trained another model for the joint system on a treebank where we replaced the gold-standard trees with trivial trees that connect the last token of each sentence to the root node, and everything in between as a left-branching chain. We dub this setting NOSYNTAX and it allows us to use exactly the same machine learning for a fair comparison between a system that has access to syntax and one without.

As the syntactic complexity in Switchboard is rather low, we compare these two systems also on a version of the WSJ where we removed all punctuation and lower-cased all words, effectively making it identical to the Switchboard setting (henceforth WSJ*). Figure 6 shows sentence boundary F_1 over training iterations when training with and without access to syntax. On both data sets, the system with access to syntax stays consistently above the other system. The striking difference between the data sets is that syntactic information has a much bigger impact on WSJ* than on Switchboard, which we attribute to the higher syntactic complexity of newswire text. Overall the comparison shows clearly that syntactic structure provides useful information to the task of sentence boundary detection.



(a) Switchboard



(b) WSJ*

Figure 6: The effect of syntactic information on sentence boundary prediction, on dev sets.

6 Final results

Sentence boundary detection. We optimize the number of iterations on the dev sets: for the joint model we take the iteration with the highest average between F_1 and LAS, NoSyntax is tuned according to F_1. Table 3 gives the performance of the sentence boundary detectors on test sets.[9]

On WSJ all systems are close to 98 and this high number once again affirms that the task of segmenting newspaper-quality text does not leave much space for improvement. Although the parsing models outperform MarMoT, the improvements in F_1 are not significant.

In contrast, all systems fare considerably worse on WSJ* which confirms that the orthographic clues in newspaper text suffice to segment the sentences properly. Although NoSyntax outperforms MarMoT, the difference is not significant. However, when real syntax is used (Joint) we see a huge improvement in F_1 – 10 points absolute – which is significantly better than both NoSyntax and MarMoT.

On Switchboard MarMoT is much lower and both parsing models outperform it significantly. Surprisingly the NoSyntax system achieves a very high result beating the baseline significantly by almost 4.5 points. The usage of syntax in the Joint model raises this gain to 4.8 points.

[9]We test for significance using the Wilcoxon signed-rank test with $p < 0.01$. † and ‡ denote significant increases over MarMoT and NoSyntax, respectively. * denotes significant increases over Joint (Table 4).

	WSJ	Switchboard	WSJ*
MarMoT	97.64	71.87	53.02
NoSyntax	98.21	$76.31^\dagger$	55.15
Joint	98.21	$76.65^\dagger$	$65.34^{\dagger\ddagger}$

Table 3: Sentence boundary detection results (F_1) on test sets.

Parsing. In order to evaluate the joint model on the parsing task separately we compare it to pipeline setups. We train a basic parser on single sentences using gold standard sentence boundaries, predicted POS tags and max-violation updates (Gold). The number of training iterations is tuned to optimize LAS on the dev set. This parser is used as the second stage in the pipeline models. Additionally, we also build a pipeline where we use Joint only as a sentence segmenter and then parse once again (denoted Joint-Reparsed).

Table 4 shows the results on the test sets. For WSJ, where sentence segmentation is almost trivial, we see only minor drops in LAS between Gold and the systems that use predicted sentence boundaries. Among the systems that use predicted boundaries, no differences are significant.

	WSJ	Switchboard	WSJ*
Gold	90.22	84.99	88.71
MarMoT	89.81	78.93	83.37
NoSyntax	89.95	$80.30^\dagger$	83.61
Joint	89.71	$79.97^\dagger$	$85.66^{\dagger\ddagger}$
Joint-Reparsed	89.93	$80.61^{\dagger\ddagger *}$	$85.38^{\dagger\ddagger}$

Table 4: Parsing results (LAS) on test sets for different sentence boundaries.

For WSJ* and Switchboard the picture is much different. Compared to Gold, all systems show considerable drops in accuracy which asserts that errors from the sentence boundary detection task propagate to the parser and worsen the parser accuracy. On Switchboard the parsers yield significantly better results than MarMoT. The best result is obtained after reparsing and this is also significantly better than any other system. Although there is a slight drop in accuracy between NoSyntax and Joint, this difference is not significant.

The results on WSJ* show that not only does syntax help to improve sentence segmentation, it does so to a degree that parsing results deteriorate when simpler sentence boundary detectors are used. Here, both Joint and Joint-Reparsed obtain significantly better parsing accuracies than the systems that do not have access to syntax during sentence boundary prediction. Although JOINT-REPARSED performs a bit worse, the difference compared to JOINT is not significant.

7 Related work

Zhang and Clark (2008) first showed how to train transition-based parsers with the structured perceptron (Collins, 2002) using beam search and early update (Collins and Roark, 2004). It has since become the de facto standard way of training search-based transition-based dependency parsers (Huang and Sagae, 2010; Zhang and Nivre, 2011; Bohnet et al., 2013). Huang et al. (2012) showed how max-violation leads to faster convergence for transition-based parsers and max-violation updates have subsequently been applied to other tasks such as machine translation (Yu et al., 2013) and semantic parsing (Zhao and Huang, 2015).

Sentence Boundary Detection. Sentence boundary detection has attracted only modest attention by the research community even though it is a component in every real-world NLP application. Previous work is divided into rule-based, e.g., CoreNLP (Manning et al., 2014), and machine learning approaches (e.g., OpenNLP, a re-implementation of Reynar and Ratnaparkhi (1997)'s MxTerminator). The task is often simplified to the task of period disambiguation (Kiss and Strunk, 2006), which only works on text that uses punctuation consistently. The current state of the art uses sequence labelers, e.g., a CRF (Evang et al., 2013; Dridan and Oepen, 2013). For a broad survey of methodology and tools, we refer the reader to Read et al. (2012).

Joint models. Solving several tasks jointly has lately been popular in transition-based parsing, e.g., combining parsing with POS tagging (Hatori et al., 2011; Bohnet and Nivre, 2012) and tokenization (Zhang et al., 2013b; Zhang et al., 2014). Joint approaches avoid error propagation between the subtasks and often lead to overall better models, especially for the lower level tasks that suddenly have access to syntactic information.

Our transition system is inspired by the work of Zhang et al. (2013a). They present a projective transition-based parser that jointly predicts punctuation and syntax. Their ArcEager transition system (Nivre, 2003) includes an additional transition that introduces punctuation similar to our SB transition. They also use beam search and circumvent the problem of long training sequences by chopping up the training data into pseudo-documents of at most 10 sentences. As we have shown, this solution works because the training instances are not long enough to hurt the performance. However, while this is possible for parsing, other tasks may not be able to chop up their training data.

8 Conclusion

We have demonstrated that training a structured perceptron for inexact search on very long input sequences ignores significant portions of the training data when using early update or max-violation. We then showed how this effect can be avoided by applying a different update strategy, DLASO, which leads to considerably better models in significantly less time. This effect only occurs when the training instances are very long, e.g., on whole documents, but not when training on single sentences. We also showed that the lower performance of early update and max-violation cannot be compensated for by increasing beam size or number of iterations. We compared our system for joint sentence boundary detection and dependency parsing to competitive pipeline systems showing that syntax can provide valuable information to sentence boundary prediction when punctuation and capitalization is not available.

Acknowledgments

This work was supported by the German Research Foundation (DFG) in project D8 of SFB 732. We also thank spaCy GmbH for making it possible for the third author to finish his contribution to this work.

References

Anders Björkelund and Jonas Kuhn. 2014. Learning Structured Perceptrons for Coreference Resolution with Latent Antecedents and Non-local Features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 47–57, Baltimore, Maryland, June. Association for Computational Linguistics.

Bernd Bohnet and Jonas Kuhn. 2012. The best of bothworlds – a graph-based completion model for transition-based parsers. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–87, Avignon, France, April. Association for Computational Linguistics.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465, Jeju Island, Korea, July. Association for Computational Linguistics.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. 2013. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China, August. Coling 2010 Organizing Committee.

Eugene Charniak and Mark Johnson. 2001. Edit detection and parsing for transcribed speech. In *In Proc. NAACL*, pages 118–126.

Michael Collins and Brian Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics, July.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive–aggressive algorithms. *Journal of Machine Learning Reseach*, 7:551–585, March.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: approximate large margin methods for structured prediction. In *ICML*, pages 169–176.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, pages 449–454.

Rebecca Dridan and Stephan Oepen. 2013. Document parsing: Towards realistic syntactic analysis. In *Proceedings of the 13th International Conference on Parsing Technologies*, Nara, Japan.

Kilian Evang, Valerio Basile, Grzegorz Chrupała, and Johan Bos. 2013. Elephant: Sequence labeling for word and sentence segmentation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1426, Seattle, Washington, USA, October. Association for Computational Linguistics.

Yoav Goldberg, Kai Zhao, and Liang Huang. 2013. Efficient implementation of beam-search incremental parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 628–633, Sofia, Bulgaria, August. Association for Computational Linguistics.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2011. Incremental joint pos tagging and dependency parsing in chinese. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1216–1224, Chiang Mai, Thailand, November. Asian Federation of Natural Language Processing.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden, July. Association for Computational Linguistics.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured Perceptron with Inexact Search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151, Montréal, Canada, June. Association for Computational Linguistics.

Tibor Kiss and Jan Strunk. 2006. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525.

Jáchym Kolář, Elizabeth Shriberg, and Yang Liu. 2006. Using prosody for automatic sentence segmentation of multi-party meetings. In *Text, Speech and Dialogue*, pages 629–636. Springer.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

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

Thomas Müller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA, October. Association for Computational Linguistics.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76, Paris, France, October. Association for Computational Linguistics.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August. Association for Computational Linguistics.

Jonathon Read, Rebecca Dridan, Stephan Oepen, and Lars Jørgen Solberg. 2012. Sentence boundary detection: A long solved problem? In *Proceedings of COLING 2012: Posters*, pages 985–994, Mumbai, India, December. The COLING 2012 Organizing Committee.

Jeffrey C. Reynar and Adwait Ratnaparkhi. 1997. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 16–19, Washington, DC, USA, March. Association for Computational Linguistics.

Heng Yu, Liang Huang, Haitao Mi, and Kai Zhao. 2013. Max-violation perceptron and forced decoding for scalable MT training. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1112–1123, Seattle, Washington, USA, October. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii, October. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.

Dongdong Zhang, Shuangzhi Wu, Nan Yang, and Mu Li. 2013a. Punctuation prediction with transition-based parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 752–760, Sofia, Bulgaria, August. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2013b. Chinese parsing exploiting characters. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 125–134, Sofia, Bulgaria, August. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2014. Character-level chinese dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1326–1336, Baltimore, Maryland, June. Association for Computational Linguistics.

Kai Zhao and Liang Huang. 2015. Type-driven incremental semantic parsing with polymorphism. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1416–1421, Denver, Colorado, May–June. Association for Computational Linguistics.