Extending a Parser to Distant Domains Using a Few Dozen Partially Annotated Examples

Vidur Joshi, Matthew Peters, Mark Hopkins
Allen Institute for AI, Seattle, WA
{vidurj, matthewp, markh}@allenai.org

Abstract

We revisit domain adaptation for parsers in the neural era. First we show that recent advances in word representations greatly diminish the need for domain adaptation when the target domain is syntactically similar to the source domain. As evidence, we train a parser on the Wall Street Journal alone that achieves over 90% F_1 on the Brown corpus. For more syntactically distant domains, we provide a simple way to adapt a parser using only dozens of partial annotations. For instance, we increase the percentage of error-free geometry-domain parses in a held-out set from 45% to 73% using approximately five dozen training examples. In the process, we demonstrate a new state-of-the-art single model result on the Wall Street Journal test set of 94.3%. This is an absolute increase of 1.7% over the previous state-of-the-art of 92.6%.

1 Introduction

Statistical parsers are often criticized for their performance outside of the domain they were trained on. The most straightforward remedy would be more training data in the target domain, but building treebanks (Marcus et al., 1993) is expensive.

In this paper, we revisit this issue in light of recent developments in neural natural language processing. Our paper rests on two observations:

1. **It is trivial to train on partial annotations using a span-focused model.** Stern et al. (2017a) demonstrated that a parser with minimal dependence between the decisions that produce a parse can achieve state-of-the-art performance. We modify their parser, henceforth MSP, so that it trains directly on individual labeled spans instead of parse trees. This results in a parser that can be trained, with no adjustments to the training regime, from partial sentence bracketings.

2. **The use of contextualized word representations (Peters et al., 2017; McCann et al., 2017) greatly reduces the amount of data needed to train linguistic models.** Contextualized word representations, which encode tokens conditioned on their context in a sentence, have been shown to give significant boosts across a variety of NLP tasks, and also to reduce the amount of data needed by an order of magnitude in some tasks.

Given [the circle [at the right] with [designated center, designated perpendicular, and radius 5]] .

In [the figure above] , [[AD = 4] , [AB = 3] and [CD = 9]] .

[Diameter AC] is perpendicular [to chord BD] [at E] .

Figure 1: An example of partial annotations. Annotators indicate that a span is a constituent by enclosing it in square brackets.

Taken together, this suggests a way to rapidly extend a newswire-trained parser to new domains. Specifically, we will show it is possible to achieve large out-of-domain performance improvements using only dozens of partially annotated sentences, like those shown in Figure 1. The resulting parser also does not suffer any degradation on the newswire domain.

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1190–1199
Melbourne, Australia, July 15 - 20, 2018.

Along the way, we provide several other notable contributions:

- We raise the state-of-the-art single-model F_1-score for constituency parsing from 92.6% to 94.3% on the Wall Street Journal (WSJ) test set. A trained model is publicly available.[1]
- We show that, even without domain-specific training data, our parser has much less out-of-domain degradation than previous parsers on "newswire-adjacent" domains like the Brown corpus.
- We provide a version of MSP which predicts its own POS tags (rather than requiring a third-party tagger).

2 The Reconciled Span Parser (RSP)

When we allow annotators to selectively annotate important phenomena, we make the process faster and simpler (Mielens et al., 2015). Unfortunately, this produces a disconnect between the model (which typically asserts the probability of a full parse tree) and the annotation task (which asserts the correctness of some subcomponent, like a constituent span or a dependency arc). There is a body of research (Hwa, 1999; Li et al., 2016) that discusses how to bridge this gap by modifying the training data, training algorithm, or the training objective.

Alternatively, we could just better align the model with the annotation task. Specifically, we could train a parser whose base model predicts exactly what we ask the annotator to annotate, e.g. whether a particular span is a constituent. This makes it trivial to train with partial or full annotations, because the training data reduces to a collection of span labels in either case.

Luckily, recent state-of-the-art results that model NLP tasks as independently classified spans (Stern et al., 2017a) suggest this strategy is currently viable. In this section, we present the Reconciled Span Parser (RSP), a modified version of the Minimal Span Parser (MSP) of Stern et al. (2017a). RSP differs from MSP in the following ways:

- **It is trained on a span classification task.** MSP trains on a maximum margin objective; that is, the loss function penalizes the violation of a margin between the scores of the gold parse and the next highest scoring parse decoded. This couples its training procedure with its decoding procedure, resulting in two versions, a top-down parser and a chart parser. To allow our model to be trained on partial annotations, we change the training task to be the span classification task described below.
- **It uses contextualized word representations instead of predicted part-of-speech tags.** Our model uses contextualized word representations as described in Peters et al. (2018). It does not take part-of-speech-tags as input, eliminating the dependence of the parser on a newswire-trained POS-tagger.

2.1 Overview

We will view a parse tree as a labeling of all the spans of a sentence such that:

- Every constituent span is labeled with the sequence of non-terminals assigned to it in the parse tree. For instance, span $(2, 4)$ in Figure 2b is labeled with the sequence $\langle S, \text{VP}\rangle$, as shown in Figure 2a.
- Every non-constituent is labeled with the empty sequence.

Given a sentence represented by a sequence of tokens x of length n, define $\mathsf{spans}(x) = \{(i, j) \mid 0 \leq i < j \leq n\}$. Define a *parse* for sentence x as a function $\pi : \mathsf{spans}(x) \mapsto \mathcal{L}$ where $\mathcal{L}$ is the set of all sequences of non-terminal tags, including the empty sequence.

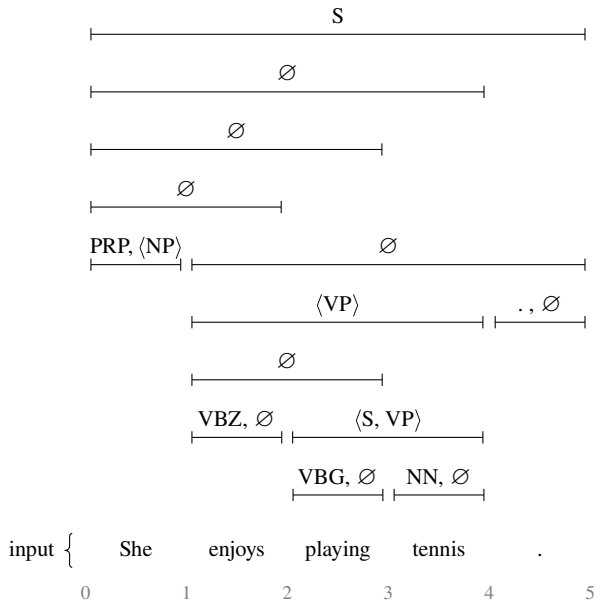
We model the probability of a parse as the independent product of its span labels:

$$\begin{aligned} Pr(\pi|x) &= \prod_{s \in \mathsf{spans}(x)} Pr(\pi(s) \mid x, s) \\ \Rightarrow \log Pr(\pi|x) &= \sum_{s \in \mathsf{spans}(x)} \log Pr(\pi(s) \mid x, s) \end{aligned}$$

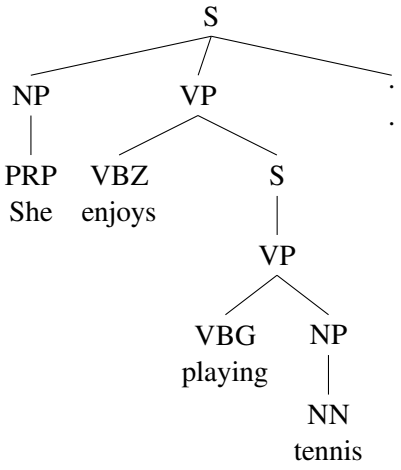
Hence, we will train a base model $\sigma(l \mid x, s)$ to estimate the log probability of label l for span s (given sentence x), and we will score the overall parse with:

$$\mathsf{score}(\pi|x) = \sum_{s \in \mathsf{spans}(x)} \sigma(\pi(s) \mid x, s)$$

[1] http://allennlp.org/models



(a) Spans classified by the parsing procedure. Note that leaves have their part-of-speech tags predicted in addition to their sequence of non-terminals.



(b) The resulting parse tree.

Figure 2: The correspondence between labeled spans and a parse tree. This diagram is adapted from figure 1 in (Stern et al., 2017a).

Note that this probability model accords mass to mis-structured trees (e.g. overlapping spans like (2, 5) and (3, 7) cannot both be constituents of a well-formed tree). We solve the following Integer Linear Program (ILP)[2] to find the highest scoring parse that admits a well-formed tree:

$$\max_{\delta} \sum_{(i,j) \in \mathsf{spans}(x)} v^{+}_{(i,j)} \delta_{(i,j)} + v^{-}_{(i,j)} (1 - \delta_{(i,j)})$$

subject to:

$$\begin{aligned} i < k < j < m \quad &\implies \quad \delta_{(i,j)} + \delta_{(k,m)} \leq 1 \\ (i,j) \in \mathsf{spans}(x) \quad &\implies \quad \delta_{(i,j)} \in \{0, 1\} \end{aligned}$$

where:

$$\begin{aligned} v^{+}_{(i,j)} &= \max_{l \text{ s.t. } l \neq \varnothing} \sigma(l \mid x, (i,j)) \\ v^{-}_{(i,j)} &= \sigma(\varnothing \mid x, (i,j)) \end{aligned}$$

[2]There are a number of ways to reconcile the span conflicts, including an adaptation of the standard dynamic programming chart parsing algorithm to work with spans of an unbinarized tree. However it turns out that the classification model rarely produces span conflicts, so all methods we tried performed equivalently well.

2.2 Classification Model

For our span classification model $\sigma(l \mid x, s)$, we use the model from (Stern et al., 2017a), which leverages a method for encoding spans from (Wang and Chang, 2016; Cross and Huang, 2016). First, it creates a sentence encoding by running a two-layer bidirectional LSTM over the sentence to obtain forward and backward encodings for each position i, denoted by $\boldsymbol{f}_i$ and $\boldsymbol{b}_i$ respectively. Then, spans are encoded by the difference in LSTM states immediately before and after the span; that is, span (i, j) is encoded as the concatenation of the vector differences $\boldsymbol{f}_j - \boldsymbol{f}_{i-1}$ and $\boldsymbol{b}_i - \boldsymbol{b}_{j+1}$. A one-layer feedforward network maps each span representation to a distribution over labels.

Classification Model Parameters and Initializations

We preserve the settings used in Stern et al. (2017a) where possible. As a result, the size of the hidden dimensions of the LSTM and the feedforward network is 250. The dropout ratio for the LSTM is set to 0.4 . Unlike the model it is based on, our model uses word embeddings of length 1124. These result from concatenating a 100 dimension learned word embedding, with a 1024 di-

Parser	Rec	Prec	F_1
RNNG (Dyer et al., 2016)	-	-	91.7
MSP (Stern et al., 2017a)	90.6	93.0	91.8
(Stern et al., 2017b)	92.6	92.6	92.6
RSP	93.8	94.8	94.3

Table 1: Parsing performance on WSJTEST, along with the results of other recent single-model parsers trained without external parse data.

	Recall	Precision	F1
all features	94.20	94.77	94.48
–ELMo	91.63	93.05	92.34

Table 2: Feature ablation on WSJDEV.

mension learned linear combination of the internal states of a bidirectional language model run on the input sentence as described in Peters et al. (2018). We refer to them below as ELMo (Embeddings from Language Models). For the learned embeddings, words with n occurrences in the training data are replaced by $\langle$UNK$\rangle$ with probability $\frac{1+\frac{n}{10}}{1+n}$. This does not affect the ELMo component of the word embeddings. As a result, even common words are replaced with probability at least $\frac{1}{10}$, making the model rely on the ELMo embeddings instead of the learned embeddings. To make the model self-contained, it does not take part-of-speech tags as input. Using a linear layer over the last hidden layer of the classification model, part-of-speech tags are predicted for spans containing single words.

3 Analysis of RSP

3.1 Performance on Newswire

On WSJTEST[3], RSP outperforms (see Table 1) all previous single models trained on WSJTRAIN by a significant margin, raising the state-of-the-art result from 92.6% to 94.3%. Additionally, our predicted part-of-speech tags achieve 97.72%[4] accuracy on WSJTEST.

[3]For all our experiments on the WSJ component of the Penn Treebank (Marcus et al., 1993), we use the standard split which is sections 2-21 for training, henceforth WSJTRAIN, section 22 for development, henceforth WSJDEV, and 23 for testing, henceforth WSJTEST.

[4]The split we used is not standard for part-of-speech tagging. As a result, we do not compare to part-of-speech taggers.

3.2 Beyond Newswire

The Brown Corpus

The Brown corpus (Marcus et al., 1993) is a standard benchmark used to assess WSJ-trained parsers outside of the newswire domain. When (Kummerfeld et al., 2012) parsed the various Brown verticals with the (then state-of-the-art) Charniak parser (Charniak, 2000; Charniak and Johnson, 2005; McClosky et al., 2006a), it achieved F_1 scores between 83% and 86%, even though its F_1 score on WSJTEST was 92.1%.

In Table 3, we discover that RSP does not suffer nearly as much degradation, with an average F_1-score of 90.3%. To determine whether this increased portability is because of the parser architecture or the use of ELMo vectors, we also run MSP on the Brown verticals. We used the Stanford tagger[5] (Toutanova et al., 2003) to tag WSJTRAIN and the Brown verticals so that MSP could be given these at train and test time. We learned that most of the improvement can be attributed to the ELMo word representations. In fact, even if we use MSP with gold POS tags, the average performance is 3.4% below RSP.

Question Bank and Genia

Despite being a standard benchmark for parsing domain adaptation, the Brown corpus has considerable commonality with newswire text. It is primarily composed of well-formed sentences with similar syntactic phenomena. Perhaps the main challenge with the Brown corpus is a difference in vocabulary, rather than a difference in syntax, which may explain the success of RSP, which leverages contextualized embeddings learned from a large corpus.

If we try to run RSP on a more syntactically divergent corpus like QuestionBank[6] (Judge et al., 2006), we find much more performance degradation. This is unsurprising, since WSJTRAIN does not contain many examples of question syntax. But how many examples do we need, to get good performance?

[5]We used the english-left3words-distsim.tagger model from the 2017-06-09 release of the Stanford POS tagger since it achieved the best accuracy on the Brown corpus.

[6]For all our experiments on QuestionBank, we use the following split: sentences 1-1000 and 2001-3000 for training, henceforth QBANKTRAIN, 1001-1500 and 3001-3500 for development, henceforth QBANKDEV, and 1501-2000 and 2501-4000 for testing, henceforth QBANKTEST. This split is described at https://nlp.stanford.edu/data/QuestionBank-Stanford.shtml.

Section	F_1			
	RSP	**MSP + Stanford POS tags**	**MSP + gold POS tags**	**Charniak**
F (popular)	91.42	87.01	87.84	85.91
G (biographies)	90.04	86.14	86.91	84.56
K (general)	90.08	85.53	86.46	84.09
L (mystery)	89.65	85.61	86.47	83.95
M (science)	90.52	86.91	87.52	84.65
N (adventure)	91.00	86.53	87.53	85.2
P (romance)	89.76	85.77	86.59	84.09
R (humor)	89.54	84.98	85.69	83.60
average	90.25	86.06	86.88	84.51

Table 3: Parsing performance on Brown verticals. MSP refers to the Minimal Span Parser (Stern et al., 2017a). Charniak refers to the Charniak parser with reranking and self-training (Charniak, 2000; Charniak and Johnson, 2005; McClosky et al., 2006a). MSP + Stanford POS tags refers to MSP trained and tested using part-of-speech tags predicted by the Stanford tagger (Toutanova et al., 2003).

Training Data		Rec.	Prec.	F_1
WSJ	QBANK			
40k	0	91.07	88.77	89.91
0	2k	94.44	96.23	95.32
40k	2k	95.84	97.02	96.43
40k	50	93.85	95.91	94.87
40k	100	95.08	96.06	95.57
40k	400	94.94	97.05	95.99

Table 4: Performance of RSP on QBANKDEV.

Training Data		Rec	Prec	F_1
WSJ	GENIA			
40k	0	72.51	88.84	79.85
0k	14k	88.04	92.30	90.12
40k	14k	88.24	92.33	90.24
40k	50	82.30	90.55	86.23
40k	100	83.94	89.97	86.85
40k	500	85.52	91.01	88.18

Table 5: Performance of RSP on GENIADEV.

For the experiments summarized in table 4 and table 5 involving 40k sentences from WSJTRAIN, we started with RSP trained on WSJTRAIN, and fine-tuned it on minibatches containing an equal number of target domain and WSJTRAIN sentences.

Surprisingly, with only 50 annotated questions (see Table 4), performance on QBANKDEV jumps 5 points, from 89.9% to 94.9%. This is only 1.5% below training with all of WSJTRAIN and QBANKTRAIN. The resulting system improves slightly on WSJTEST getting 94.38%.

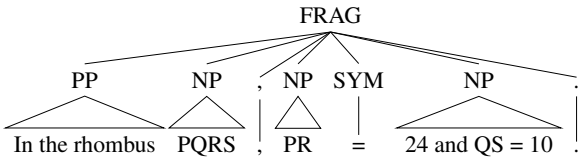
On the more difficult GENIA corpus of biomedical abstracts (Tateisi et al., 2005), we see a similar, if somewhat less dramatic, trend. See Table 5. With 50 annotated sentences, performance on GENIADEV jumps from 79.5% to 86.2%, outperforming all but one parser from David McClosky's thesis (McClosky, 2010) – the one that trains on all 14k sentences from GENIATRAIN and self-trains using 270k sentences from PubMed. That parser achieves 87.6%, which we outperform with just 500 sentences from GENIATRAIN.

These results suggest that it is currently feasible to extend a parser to a syntactically distant domain (for which no gold parses exist) with a couple hours of effort. We explore this possibility in the next section.

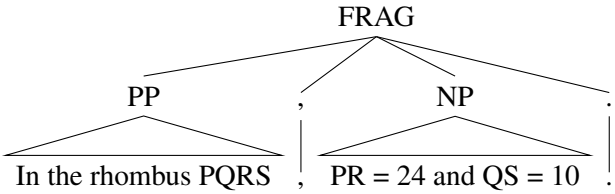
4 Rapid Parser Extension

To create a parser for their geometry question answering system, (Seo et al., 2015) did the following:

- Designed regular expressions to identify mathematical expressions.
- Replaced the identified expressions with dummy words.
- Parsed the resulting sentences.



(a) Training only on WSJTRAIN.



(b) Retraining on partial annotations.

Figure 3: The top-level split for the development sentence "In the rhombus PQRS, PR = 24 and QS = 10." before and after retraining RSP on 63 partially annotated geometry statements.

- Substituted the regex-analyzed expressions for the dummy words in the parses.

It is clear why this was necessary. Figure 3 (top) shows how RSP (trained only on WSJTRAIN) parses the sentence "In the rhombus PQRS, PR = 24 and QS = 10." The result is completely wrong, and useless to a downstream application.

Still, beyond just the inconvenience of building additional infrastructure, there are downsides to the "regex-and-replace" strategy:

1. **It assumes that each expression always maps to the same constituent label.** Consider "$2x = 3y$". This is a verb phrase in the sentence "In the above figure, x is prime and $2x = 3y$." However, it is a noun phrase in the sentence "The equation $2x = 3y$ has 2 solutions." If we replace both instances with the same dummy word, the parser will almost certainly become confused in one of the two instances.

2. **It assumes that each expression is always a constituent.** Suppose that we replace the expression "$AB < 30$" with a dummy word. This means we cannot properly parse a sentence like "When angle $AB < 30$, the lines are parallel," because the constituent "angle AB" no longer exists in the resulting sentence.

3. **It does not handle other syntactic variation.** As we will see in the next section, the geometry domain has a propensity for using right-attaching participial adjective phrases, like "labeled x" in the phrase "the segment labeled x." Encouraging a parser to recognize this syntactic construct is out-of-scope for the "regex-and-replace" strategy.

Instead, we propose directly extending the parser by providing a few domain-specific examples like those in Figure 1. Because RSP's model directly predicts span constituency, we can simply mark up a sentence with the "tricky" domain-specific constituents that the model will not already have learned from WSJTRAIN. For instance, we mark up NOUN-LABEL constructs like "chord BD", and equations like "AD = 4".

From these marked-up sentences, we can extract training instances declaring the constituency of certain spans (like "to chord BD" in the third example) and the implied non-constituency of certain spans (like "perpendicular to chord" in the third example). We also allow annotators to explicitly declare the non-constituency of a span via an alternative markup (not shown).

We do not require annotators to provide span labels (although they can if desired). If a training instance merely declares a span to be a constituent (but does not provide a particular label), then the loss function only records loss when that span is classified as a non-constituent (i.e. any label is ok).

5 Experiments

5.1 Geometry Questions

We took the publicly available training data from (Seo et al., 2015), split the data into sentences, and then annotated each sentence as in Figure 1. Next, we randomly split these sentences into GEOTRAIN and GEODEV[7]. After removing duplicate sentences spanning both sets, we ended up with 63 annotated sentences in GEOTRAIN and 62 in GEODEV. In GEOTRAIN, we made an average of 2.8 constituent declarations and 0.3 (explicit) non-constituent declarations per sentence.

After preparing the data, we started with RSP trained on WSJTRAIN, and fine-tuned it on mini-batches containing 50 randomly selected WSJTRAIN sentences, plus all of GEOTRAIN. The results are in table 6. After fine-tuning, the model

[7] GEOTRAIN and GEODEV are available at https://github.com/vidurj/parser-adaptation/tree/master/data.

Training Data	GEODEV		WSJTEST
	correct constituents %	error-free %	F_1
WSJTRAIN	71.9	45.2	94.28
WSJTRAIN + GEOTRAIN	87.0	72.6	94.30

Table 6: RSP performance on GEODEV.

Training Data	BIOCHEMDEV		WSJTEST
	correct constituents %	error-free %	F_1
WSJTRAIN	70.1	27.0	94.28
WSJTRAIN + BIOCHEMTRAIN	79.5	46.7	94.23

Table 7: RSP performance on BIOCHEMDEV.

- **Given [a circle with [the tangent shown]] .**
- **Find the hypotenuse of [the triangle labeled t] .**
- **Examine [the following diagram with [the square highlighted]] .**

Figure 4: Three partial annotations targeting right-attaching participial adjectives.

gets 87% of the 185 annotations on GEODEV correct, compared with 71.9% before fine-tuning[8]. Moreover, the fraction of sentences with no errors increases from 45.2% to 72.6%. With only a few dozen partially-annotated training examples, not only do we see a large increase in domain performance, but there is also no degradation in the parser's performance on newswire. Some GEODEV parses have enormous qualitative differences, like the example shown in Figure 3.

For the GEODEV sentences on which we get errors after retraining, the errors fall predominantly into three categories. First, approximately 44% have some mishandled math syntax, like failing to recognize "dimensions 16 by 8" as a constituent, or providing a flat structuring of the equation "BAC = 1/4 * ACB" (instead of recognizing "1/4 * ACB" as a subconstituent). Second, approximately 19% have PP-attachment errors. Third, another 19% fail to correctly analyze right-attaching participial adjectives like "labeled x" in the noun phrase "the segment labeled x" or "indicated" in the noun phrase "the center indicated." This phenomenon is unusually frequent in geometry but was insufficiently marked-up in our training examples. For instance, while we have a training instance "Find [the measure of [the angle designated by x]]," it does not explicitly highlight the constituency of "designated by x". This suggests that in practice, this domain adaptation method could benefit from an iterative cycle in which a user assesses the parser's errors on their target domain, creates some partial annotations that address these issues, retrains the parser, and then repeats the process until satisfied. As a proof-of-concept, we invented 3 additional sentences with right-attaching participial adjectives (shown in Figure 4), added them to GEOTRAIN, and then retrained. Indeed, the handling of participial adjectives in GEODEV improved, increasing the overall percentage of correctly identified constituents to 88.6% and the percentage of error-free sentences to 75.8%.

5.2 Biomedicine and Chemistry

We ran a similar experiment using biomedical and chemistry text, taken from the unannotated data provided by (Nivre et al., 2007). We partially annotated 134 sentences and randomly split them into BIOCHEMTRAIN (72 sentences) and BIOCHEMDEV (62 sentences)[9]. In BIOCHEMTRAIN, we made an average of 4.2 constituent declarations per sentence. We made no non-constituent declarations.

Again, we started with RSP trained on WSJTRAIN, and fine-tuned it on minibatches containing annotations from 50 randomly selected WSJ-

[8] This improvement has a p-value of 10^{-4} under the one-sided, two-sample difference between proportions test.

[9] BIOCHEMTRAIN and BIOCHEMDEV are available at https://github.com/vidurj/parser-adaptation/tree/master/data.

TRAIN sentences, plus all of BIOCHEMTRAIN. Table 7 shows the improvement in the percentage of correctly-identified annotated constituents and the percentage of test sentences for which the parse agrees with every annotation. As with the geometry domain, we get significant improvements using only dozens of partially annotated training sentences.

6 Related Work

The two major themes of this paper, domain adaptation and learning from partial annotation, each have a long tradition in natural language processing.

6.1 Domain Adaptation

Domain adaptation has been recognized as a major NLP problem for over a decade (Ben-David et al., 2006; Blitzer et al., 2006; Daumé, 2007; Finkel and Manning, 2009). In particular, domain adaptation for parsers (Plank, 2011; Ma and Xia, 2013) has received considerable attention. Much of this work (McClosky et al., 2006b; Reichart and Rappoport, 2007; Sagae and Tsujii, 2007; Kawahara and Uchimoto, 2008; McClosky et al., 2010; Sagae, 2010; Baucom et al., 2013; Yu et al., 2015) has focused on how to best use co-training (Blum and Mitchell, 1998) or self-training to augment a small domain corpus, or how to best combine models to perform well on a particular domain.

In this work, we focus on the direct impact that just a few dozen partially annotated out-of-domain examples can have, when using a particular neural model with contextualized word representations. Co-training, self-training, and model combination are orthogonal to our approach. Our work is a spiritual successor to (Garrette and Baldridge, 2013), which shows how to train a part-of-speech tagger with a minimal amount of annotation effort.

6.2 Learning from Partial Annotation

Most literature on training parsers from partial annotations (Sassano and Kurohashi, 2010; Spreyer et al., 2010; Flannery et al., 2011; Flannery and Mori, 2015; Mielens et al., 2015) focuses on dependency parsing. (Li et al., 2016) provides a good overview. Here we highlight three important high-level strategies.

The first is "complete-then-train" (Mirroshandel and Nasr, 2011; Majidi and Crane, 2013), which "completes" every partially annotated dependency parse by finding the most likely parse (according to an already trained parser model) that respects the constraints of the partial annotations. These "completed" parses are then used to train a new parser.

The second strategy (Nivre et al., 2014; Li et al., 2016) is similar to "complete-then-train," but integrates parse completion into the training process. At each iteration, new "complete" parses are created using the parser model from the most recent training iteration.

The third strategy (Li et al., 2014, 2016) transforms each partial annotation into a forest of parses that encodes all fully-specified parses permitted by the partial annotation. Then, the training objective is modified to support optimization over these forests.

Our work differs from these in two respects. First, since we are training a constituency parser, our partial annotations are constituent bracketings rather than dependency arcs. Second, and more importantly, we can use the partial annotations for training without modifying either the training algorithm or the training data.

While the bulk of the literature on training from partial annotations focuses on dependency parsing, the earliest papers (Pereira and Schabes, 1992; Hwa, 1999) focus on constituency parsing. These leverage an adapted version of the inside-outside algorithm for estimating the parameters of a probabilistic context-free grammar (PCFG). Our work is not tied to PCFG parsing, nor does it require a specialized training algorithm when going from full annotations to partial annotations.

7 Conclusion

Recent developments in neural natural language processing have made it very easy to build custom parsers. Not only do contextualized word representations help parsers learn the syntax of new domains with very few examples, but they also work extremely well with parsing models that correspond directly with a granular and intuitive annotation task (like identifying whether a span is a constituent). This allows you to train with either full or partial annotations without any change to the training process.

This work provides a convenient path forward for the researcher who requires a parser for their domain, but laments that "parsers don't work outside of newswire." With a couple hours of effort

(and a layman's understanding of syntactic building blocks), they can get significant performance improvements. We envision an iterative use case in which a user assesses a parser's errors on their target domain, creates some partial annotations to teach the parser how to fix these errors, then retrains the parser, repeating the process until they are satisfied.

References

Eric Baucom, Levi King, and Sandra Kübler. 2013. Domain adaptation for parsing. In *RANLP*.

Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. 2006. Analysis of representations for domain adaptation. In *NIPS*.

John Blitzer, Ryan T. McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspondence learning. In *EMNLP*.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*. ACM, pages 92–100.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *ANLP*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pages 173–180.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *EMNLP*.

Hal Daumé. 2007. Frustratingly easy domain adaptation. *CoRR* abs/0907.1815.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. *CoRR* abs/1602.07776. http://arxiv.org/abs/1602.07776.

Jenny Rose Finkel and Christopher D. Manning. 2009. Hierarchical bayesian domain adaptation. In *HLT-NAACL*.

Daniel Flannery, Yusuke Miyao, Graham Neubig, and Shinsuke Mori. 2011. Training dependency parsers from partially annotated corpora. In *IJCNLP*.

Daniel Flannery and Shinsuke Mori. 2015. Combining active learning and partial annotation for domain adaptation of a japanese dependency parser. In *IWPT*.

Dan Garrette and Jason Baldridge. 2013. Learning a part-of-speech tagger from two hours of annotation. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. pages 138–147.

Rebecca Hwa. 1999. Supervised grammar induction using training data with limited constituent information. *CoRR* cs.CL/9905001.

John Judge, Aoife Cahill, and Josef Van Genabith. 2006. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 497–504.

Daisuke Kawahara and Kiyotaka Uchimoto. 2008. Learning reliability of parses for domain adaptation of dependency parsing. In *IJCNLP*.

Jonathan K. Kummerfeld, David Leo Wright Hall, James R. Curran, and Dan Klein. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *EMNLP-CoNLL*.

Zhenghua Li, Min Zhang, and Wenliang Chen. 2014. Soft cross-lingual syntax projection for dependency parsing. In *COLING*.

Zhenghua Li, Yue Zhang, Jiayuan Chao, and Min Zhang. 2016. Training dependency parsers with partial annotation. *CoRR* abs/1609.09247.

Xuezhe Ma and Fei Xia. 2013. Dependency parser adaptation with subtrees from auto-parsed target domain data. In *ACL*.

Saeed Majidi and Gregory R. Crane. 2013. Committee-based active learning for dependency parsing. In *TPDL*.

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

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *NIPS*.

David McClosky. 2010. Any domain parsing: automatic domain adaptation for natural language parsing .

David McClosky, Eugene Charniak, and Mark Johnson. 2006a. Effective self-training for parsing. In *HLT-NAACL*.

David McClosky, Eugene Charniak, and Mark Johnson. 2006b. Reranking and self-training for parser adaptation. In *ACL*.

David McClosky, Eugene Charniak, and Mark Johnson. 2010. Automatic domain adaptation for parsing. In *HLT-NAACL*.

Jason Mielens, Liang Sun, and Jason Baldridge. 2015. Parse imputation for dependency annotations. In *ACL*.

Seyed Abolghasem Mirroshandel and Alexis Nasr. 2011. Active learning for dependency parsing using partially annotated sentences. In *IWPT*.

Joakim Nivre, Yoav Goldberg, and Ryan T. McDonald. 2014. Constrained arc-eager dependency parsing. *Computational Linguistics* 40:249–527.

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 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Fernando Pereira and Yves Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. In *ACL*.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. 2018. Deep contextualized word representations. *ArXiv e-prints* .

Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *ACL*.

Barbara Plank. 2011. *Domain adaptation for parsing*. Citeseer.

Roi Reichart and Ari Rappoport. 2007. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. In *ACL*.

Kenji Sagae. 2010. Self-training without reranking for parser domain adaptation and its impact on semantic role labeling.

Kenji Sagae and Jun'ichi Tsujii. 2007. Dependency parsing and domain adaptation with lr models and parser ensembles. In *EMNLP-CoNLL*.

Manabu Sassano and Sadao Kurohashi. 2010. Using smaller constituents rather than sentences in active learning for japanese dependency parsing. In *ACL*.

Min Joon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *EMNLP*.

Kathrin Spreyer, Lilja Ovrelid, and Jonas Kuhn. 2010. Training parsers on partial trees: A cross-language comparison. In *LREC*.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017a. A minimal span-based neural constituency parser. *CoRR* abs/1705.03919. http://arxiv.org/abs/1705.03919.

Mitchell Stern, Daniel Fried, and Dan Klein. 2017b. Effective inference for generative neural parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. pages 1695–1700. https://aclanthology.info/papers/D17-1178/d17-1178.

Yuka Tateisi, Akane Yakushiji, Tomoko Ohta, and Jun'ichi Tsujii. 2005. Syntax annotation for the genia corpus. In *Companion Volume to the Proceedings of Conference including Posters/Demos and tutorial abstracts*.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2003, Edmonton, Canada, May 27 - June 1, 2003*. http://aclweb.org/anthology/N/N03/N03-1033.pdf.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional lstm. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 2306–2315.

Juntao Yu, Mohab Elkaref, and Bernd Bohnet. 2015. Domain adaptation for dependency parsing via self-training. In *IWPT*.