Semi-supervised dependency parsing using generalized tri-training

Anders Søgaard and Christian Rishøj
Center for Language Technology
University of Copenhagen
{soegaard|crjensen}@hum.ku.dk

Abstract

Martins et al. (2008) presented what to the best of our knowledge still ranks as the best overall result on the CONLL-X Shared Task datasets. The paper shows how triads of stacked dependency parsers described in Martins et al. (2008) can label unlabeled data for each other in a way similar to co-training and produce end parsers that are significantly better than any of the stacked input parsers. We evaluate our system on five datasets from the CONLL-X Shared Task and obtain 10–20% error reductions, incl. the best reported results on four of them. We compare our approach to other semi-supervised learning algorithms.

1 Introduction

Semi-supervised learning of structured variables is a difficult problem that has received considerable attention recently, but most results have been negative (Abney, 2008). This paper uses stacked learning (Wolpert, 1992) to reduce structured variables, i.e. dependency graphs, to multinomial variables, i.e. attachment and labeling decisions, which are easier to manage in semi-supervised learning scenarios, and which can later be combined into dependency trees using parsing algorithms for arc-factored dependency parsing. Our approach thus combines ensemble-based methods and semi-supervised learning.

Ensemble-based methods such as stacked learning are used to reduce the instability of classifiers, to average out their errors and to combine the strengths of diverse learning algorithms. Ensemble-based methods have attracted a lot of attention in dependency parsing recently (Sagae and Lavie, 2006; Hall et al., 2007; Nivre and McDonald, 2008; Martins et al., 2008; Fishel and Nivre, 2009; Surdeanu and Manning, 2010). Nivre and McDonald (2008) were first to introduce stacking in the context of dependency parsing.

Semi-supervised learning is typically motivated by data sparseness. For many classification tasks in natural language processing, labeled data can be in short supply but unlabeled data is more readily available. Semi-supervised methods exploit unlabeled data in addition to labeled data to improve performance on classification tasks. If the predictions of a learner l on unlabeled data are used to improve a learner l' in semi-supervised learning, the robustness of learning will depend on the stability of l. Combining ensemble-based and semi-supervised methods may thus lead to more robust semi-supervised learning.

Ensemble-based and semi-supervised methods are some of the areas that receive most attention in machine learning today, but relatively little attention has been given to *combining* these methods (Zhou, 2009). Semi-supervised learning algorithms can be categorized with respect to the number of views, i.e. the number of feature sets, and the number of learners used to inform each other (Hady and Schwenker, 2008). Self-training and expectation maximization are perhaps the best known semi-supervised learning algorithms (Abney, 2008). They are both single-view and single-learner algorithms. Since there is thus only a single perspective on data,

Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), pages 1065–1073,
Beijing, August 2010

selecting unlabeled data points with predictions is a difficult task. There is an imminent danger that the learner amplifies its previous mistakes, and while several techniques such as balancing and throttling have been developed to avoid such caveats, using single-view and single-learner algorithms often requires both caution and experience with the modeling task at hand.

Algorithms with multiple views on data are known to be more robust. This insight led to the development of co-training (Blum and Mitchell, 1998), a two-view method where views inform each other, but it also paved the way for the integration of ensemble-based and semi-supervised methods, i.e. for methods with multiple learners. It was mentioned that relatively little work has been devoted to this topic, but there are notable exceptions:

Bennett et al. (2003) generalized boosting to semi-supervised learning in a seminal paper, where the idea of iterative or recursive ensembles was also introduced. Li and Zhou (2005) introduce *tri-training*, a form of co-training that trains an ensemble of three learners on labeled data and runs them on unlabeled data. If two learners agree on their labeling of a data point, the data point is added to the labeled data of the third learner with the prediction of the first two. Didaci and Roli (2006) extend self-training and co-training to multiple learners. Li and Zhou (2007) generalize tri-training to larger ensembles of random trees. The technique is also known as co-forests. Hady and Schwenker (2008) generalize existing ensemble-based methods for semi-supervised learning scenarios; in particular they embed ensembles in a form of co-training that is shown to maintain the diversity of the ensemble over time. Milidiu and Duarte (2009) generalize boosting at start to semi-supervised learning.

This paper applies a generalization of tri-training to two classification problems, attachment and labeling. The attachment classifier's weights are used for arc-factored dependency parsing, and the labeling classifier's weights are then used to label the dependency tree delivered by the parser.

Semi-supervised dependency parsing has attracted a lot of attention recently (Koo et al., 2008; Wang et al., 2008; Suzuki et al., 2009), but there has, to the best of our knowledge, been no previous attempts to apply tri-training or related combinations of ensemble-based and semi-supervised methods to any of these tasks, except for the work of Sagae and Tsujii (2007) discussed in Sect. 2.6. However, tri-training has been applied to Chinese chunking (Chen et al., 2006), question classification (Nguyen et al., 2008) and POS tagging (Søgaard, 2010).

We compare generalized tri-training to other semi-supervised learning algorithms, incl. self-training, the original tri-training algorithm based on bootstrap samples (Li and Zhou, 2005), co-forests (Li and Zhou, 2007) and semi-supervised support vector machines (Sindhwani and Keerthi, 2006).

Sect. 2 introduces dependency parsing and stacked learning. Stacked learning is generalized to dependency parsing, and previous work is briefly surveyed. We then describe how stacked dependency parsers can be further stacked as input for two end classifiers that can be combined to produce dependency structures. These two classifiers will learn multinomial variables (attachment and labeling) from a combination of labeled data and unlabeled data using a generalization of tri-training. Sect. 3 describes our experiments. We describe the data sets, and how the unlabeled data was prepared. Sect. 4 presents our results. Sect. 5 presents an error analysis and discusses the results in light of other results in the literature, and Sect. 6 concludes the paper.

2 Background and related work

2.1 Dependency parsing

Dependency parsing models a sentence as a tree where words are vertices and grammatical functions are directed edges (dependencies). Each word thus has a single incoming edge, except one called the root of the tree. Dependency parsing is thus a structured prediction problem with trees as structured variables. Each sentence has exponentially many possible dependency trees. Our observed variables are sentences with words labeled with part-of-speech tags. The task for

each sentence is to find the dependency tree that maximizes an objective function which in our case is learned from a combination of labeled and unlabeled data.

More formally, a dependency tree for a sentence $x = w_1, \ldots, w_n$ is a tree $T = \langle \{0, 1, \ldots, n\}, A \rangle$ with $A \subseteq V \times V$ the set of dependency arcs. Each vertex corresponds to a word in the sentence, except 0 which is the root vertex, i.e. for any $i \leq n$ $\langle i, 0 \rangle \notin A$. Since a dependency tree is a tree it is acyclic. A tree is projective if every vertex has a continuous projection, i.e. if and only if for every arc $\langle i, j \rangle \in A$ and node $k \in V$, if $i < k < j$ or $j < k < i$ then there is a subset of arcs $\{\langle i, i_1 \rangle, \langle i_1, i_2 \rangle, \ldots, \langle i_{k-1}, i_k \rangle\} \in A$ such that $i_k = k$.

In this paper we use a maximum spanning tree algorithm, the so-called Chu-Liu-Edmonds algorithm (CLE) (Edmonds, 1967) to turn the predictions of our semi-supervised classifiers into a dependency tree.

2.2 Stacked learning

Stacked generalization, or simply *stacking*, was first proposed by Wolpert (1992). Stacking is an ensemble-based learning method where multiple weak classifiers are combined in a strong end classifier. The idea is to train the end classifier directly on the predictions of the input classifiers.

Say each input classifier c_i with $1 \leq i \leq n$ receives an input $\mathbf{x}$ and outputs a prediction $c_i(\mathbf{x})$. The end classifier then takes as input $\langle \mathbf{x}, c_1(\mathbf{x}), \ldots, c_n(\mathbf{x}) \rangle$ and outputs a final prediction $c_0(\langle \mathbf{x}, c_1(\mathbf{x}), \ldots, c_n(\mathbf{x}) \rangle)$. Training is done by cross-validation. In sum, stacking is training a classifier on the output of classifiers.

2.3 Stacked dependency parsing

Stacked learning can be generalized to structured prediction tasks such as dependency parsing. Architectures for stacking dependency parsers typically only use one input parser, but otherwise the intuition is the same: the input parser is used to augment the dependency structures that the end parser is trained and evaluated on.

Nivre and McDonald (2008) first showed how the MSTParser (McDonald et al., 2005) and the MaltParser (Nivre et al., 2007) could be improved by stacking each parser on the predictions of the other. Martins et al. (2008) generalized their work, considering more combinations of parsers, and stacking the end parsers on non-local features from the predictions of the input parser, e.g. siblings and grand-parents. In this work we use three stacked dependency parsers for each language: mst2 (p_1), malt/mst2 (p_2) and malt/mst1 (p_3).

The notation "malt/mst2" means that the second-order MSTParser has been stacked on the MaltParser. The capital letters refer to feature configurations. Configuration D stacks a level 1 parser on several (non-local) features of the predictions of the level 0 parser (along with the input features): the predicted edge, siblings, grand parents and predicted head of candidate modifier if predicted edge is 0. Configuration E stacks a level 1 parser on the features in configuration D and all the predicted children of the candidate head. The chosen parser configurations are those that performed best in Martins et al. (2008) across the different datasets.

2.4 Stacking stacked dependency parsing

The input features of the input classifiers in stacked learning $\mathbf{x}$ can of course be removed from the input of the end classifier. It is also possible to stack stacked classifiers. This leaves us with four strategies for recursive stacking; namely to constantly augment the feature set, with level n classifiers trained on the predictions of the classifiers at all $n-1$ lower levels with or without the input features $\mathbf{x}$, or simply to train a level n classifier on the predictions of the level $n-1$ classifiers with or without $\mathbf{x}$.

In this work we stack stacked dependency parsers by training classifiers on the output of three stacked dependency parsers and POS tags. Consequently, we use one of the features from $\mathbf{x}$. Note that we train classifiers and not parsers on this new level 2.

The reduction is done the following way: First we train a classifier on the relative distance from a word to its head to induce attachments. For example, we may obtain the following features from the predictions of our level 1 parsers:

label	p_1	p_2	p_3	POS
1	1	-1	1	NNP
0	0	0	0	VBD

In the second row all input parsers, p_{1-3} in columnsaa 2–4, agree that the verb is the root of the sentence. Column 1 tells us that this is correct. In the first row, two out of three parsers agree on attaching the noun to the verb, which again is correct. We train level 2 classifiers on feature vectors produced this way. Note that oracle performance of the ensemble is no upper bound on the accuracy of a classifier trained on level 1 predictions this way, since a classifier may learn the right decision from three wrong predictions and a POS tag.

Second we train a classifier to predict dependency relations. Our feature vectors are similar to the ones just described, but now contain dependency label predictions, e.g.:

label	p_1	p_2	p_3	POS
SBJ	SBJ	SBJ	SBJ	NN
ROOT	ROOT	ROOT	COORD	VBN

2.5 Generalized tri-training

Tri-training was originally introduced in Li and Zhou (2005). The method involves three learners that inform each other.

Let L denote the labeled data and U the unlabeled data. Assume that three classifiers c_1, c_2, c_3 have been trained on L. In the original algorithm, the three classifiers are obtained by applying the same learning algorithm to three bootstrap samples of the labeled data; but in generalized algorithms, three different learning algorithms are used. An unlabeled datapoint in U is labeled for a classifier, say c_1, if the other two classifiers agree on its label, i.e. c_2 and c_3. Two classifiers inform the third. If the two classifiers agree on a labeling, we assume there is a good chance that they are right. In the original algorithm, learning stops when the classifiers no longer change; in generalized tri-training, a fixed stopping criterion is used. The three classifiers are combined by voting. Li and Zhou (2005) show that under certain conditions the increase in classification noise rate is compensated by the amount of newly labeled data points.

The most important condition is that the three classifiers are diverse. If the three classifiers are identical, tri-training degenerates to *self-training*. As already mentioned, Li and Zhou (2005) obtain this diversity by training classifiers on bootstrap samples. In their experiments, they consider classifiers based on decision trees, BP neural networks and naïve Bayes inference.

```
for i ∈ {1..3} do
    c_i ← train_classifier(l_i, L)
end for
repeat
    for i ∈ {1..3} do
        for x ∈ U do
            L_i ← ∅
            if c_j(x) = c_k(x)(j, k ≠ i) then
                L_i ← L_i ∪ {(x, c_j(x)}
            end if
        end for
        c_i ← train_classifier(l_i, L ∪ L_i)
    end for
until stopping criterion is met
apply c_1
```

Figure 1: Generalized tri-training.

In this paper we generalize the tri-training algorithm and use three different learning algorithms rather than bootstrap samples to create diversity: a naïve Bayes algorithm (no smoothing), random forests (Breiman, 2001) (with 100 unpruned decision trees) and an algorithm that induces unpruned decision trees. The overall algorithm is sketched in Figure 1 with l_i a learning algorithm.

Our weights are those of the random forest classifier after a fixed number of rounds. The attachment classifier iterates once over the unlabeled data, while the dependency relations classifier uses three iterations. The optimal number of iterations could of course be estimated on development data instead. Given the weights for an input sentence we use CLE to find its most likely dependency tree.

2.6 Related work

This paper uses stacking rather than voting to construct ensembles, but voting has been more

widely used in dependency parsing than stacking. Voting was first introduced in dependency parsing in Zeman and Zabokrtsky (2005). Sagae and Lavie (2006) later used weighted voting and reparsing, i.e. using CLE to find the dependency tree that reflects the maximum number of votes. They also showed that binning the vote over part-of-speech tags led to further improvements. This set-up was adopted by Hall et al. (2007) in the best performing system in the CONLL 2007 Shared Task. Fishel and Nivre (2009) later experimented with binning the vote on other features with modest improvements.

Semi-supervised dependency parsing has only recently been explored, and failures have been more frequent than successes. There are, however, noteable exceptions such as Koo et al. (2008), Wang et al. (2008), Suzuki et al. (2009) and Sagae and Gordon (2009).

The semi-supervised methods employed in these experiments are very different from more traditional scenarios such as self-training and co-training. Two approaches (Koo et al., 2008; Sagae and Gordon, 2009) use clusters obtained from large amounts of unlabeled data to augment their labeled data by introducing new features, and two approaches (Wang et al., 2008; Suzuki et al., 2009) combine probability distributions obtained from labeled data with probability distributions obtained from unlabeled data.

Successes with self-training and co-training are rare, and several authors report negative results, e.g. Spreyer and Kuhn (2009). A noteable exception in constituent-based parsing is the work of McClosky et al. (2006) who show that self-training is possible if a reranker is used to inform the underlying parser.

Sagae and Tsujii (2007) participated in (and won) the CONLL 2007 Shared Task on domain adaptation. They first trained a maximum entropy-based transition-based dependency parser on the out-of-domain labeled data and an SVM-based transition-based dependency parser on the *reversed* out-of-domain labeled data. The two parsers parse the in-domain labeled data (reversed, in the case of the SVM-based parser). Identical analyses are added to the original training set. The first parser is retrained and used to parse the test data. In sum, the authors do one round of co-training with the following selection criterion: If the two parsers produce the same dependency structures for a sentence, the dependency structure is added to the labeled data. This criterion is also the selection criterion in tri-training.

3 Experiments

3.1 Data

We use five datasets from the CONLL-X Shared Task (Buchholz and Marsi, 2006).[1] Lemmas and morphological features (FEATS) are ignored, since we only add POS and CPOS tags to unlabeled data. For German and Swedish, we use 100,000 sentences from the Leipzig Corpora Collection (Biemann et al., 2007) as unlabeled data. For Danish, Dutch, and Portuguese we use 100,000 sentences from the Europarl corpus (Koehn, 2005). The data characteristics are provided in Figure 2. The unlabeled data were POS tagged using the freely available SVMTool (Gimenez and Marquez, 2004) (model 4, left-right-left).

3.2 Algorithm

Once our data has been prepared, we train the stacked dependency parsers and use them to label training data for our classifiers ($\sim$4,000 tokens), our test data and our unlabeled data. This gives us three sets of predictions for each of the three data sets. Using the features described in Sect. 2.4 we then construct data for training our two triads of classifiers (for attachment and dependency relations). The entire architecture can be depicted as in Figure 3.

We first stack three dependency parsers as described in Martins et al. (2008). We then stack three classifiers on top of these dependency parsers (and POS tags): a naïve Bayes classifier, a random forest, and a decision tree. Finally,

[1]The CONLL-X Shared Task consists of 12 datasets, but we did not have consistently tokenized unlabeled data for Arabic, Chinese, Japanese, Slovene and Turkish. Martins et al. (2008) ignore Czech. Our experiment with the Spanish dataset crashed unexpectedly. We will post results on the website as soon as possible.

		tokens	sents	tokens/sents	POSs	DEPRELs
Danish	train	94,386	5,190	18.2	24	52
	unl (Europarl)	2,422,144	100,000	24.2	-	-
	test	5,852	322	18.2	-	-
Dutch	train	195,069	13,349	14.6	13	26
	unl (Europarl)	2,336,176	100,000	23.4	-	-
	test	5,585	386	14.5	-	-
German	train	699,610	39,216	17.8	52	46
	unl (LCC)	1,763,281	100,000	17.6	-	-
	test	5,694	357	15.9	-	-
Portuguese	train	206,678	9,071	22.3	21	55
	unl (Europarl)	2,882,967	100,000	28.8	-	-
	test	5,867	288	22.8	-	-
Swedish	train	191,467	11,042	17.4	37	56
	unl (LCC)	1,727,068	100,000	17.3	-	-
	test	5,656	389	14.5	-	-

Figure 2: Characteristics of the data sets.

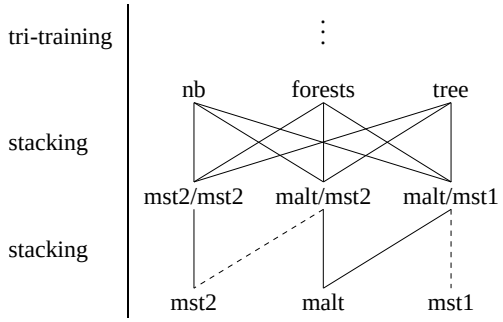

Figure 3: Tri-training stacked classifiers.

we tri-train these three stacked classifiers and for each test sentence output the weights provided by the random forest classifier. These weights are used to find the best possible dependency tree using CLE.

3.3 Baselines

The best of the stacked input parsers is of course our natural baseline.

Since we have generalized tri-training, we also compare generalized tri-training to the original tri-training algorithm based on bootstrap samples. The original tri-training algorithm is run with the same decomposition and the same features as our generalized tri-training algorithm. We use the learning algorithm originally used in Li and Zhou (2005), namely C4.5. We also compare our results to self-training (no pool, no growth rate) and co-forests (Li and Zhou, 2007). Finally, we compare our results to semi-supervised support vector machines (S3VMs) (Sindhwani and Keerthi, 2006). Since S3VMs produce binary classifiers, and one-vs.-many combination would be very time-consuming, we train a binary classifier that produces a probability that any candidate arc is correct and do greedy head selection. We optimized the feature set and included a total of seven features (head POS, dependent POS, dependent left neighbor POS, distance+direction, predictions of the three classifiers).

4 Results

Our results are presented in Figure 4. Labeled (LAS) and unlabeled attachment scores (UAS) and labeling accuracy (LA) are defined as usual and include punctuation signs unless otherwise noted. Difference (Δ) in LAS, error reduction and p-value compare our results to the best input stacked parser (malt/mst2, excerpt for Swedish).

Generalized tri-training (tri-training-CLE), i.e. using CLE to find the best well-formed dependency trees given the weights provided by our tri-trained random forest classifier, leads to highly significant improvements on *all* data sets ($p < 0.001$) with an average error reduction of 14,9%. The results for the other semi-supervised learning algorithms are presented in Figure 5. We only used 10% of the unlabeled data (10k sentences) in this experiment and only did unlabeled parsing, but it is quite evident that these learning strategies seem less promising than gen-

Danish	LAS(%)	UAS(%)	LA(%)	EM(%)	Δ LAS	err.red(%)	p-value
mst2	84.64	89.11	91.35	24.84			
malt/mst2	86.36	90.50	92.09	27.64			
malt/mst1	86.11	90.23	91.87	25.78			
tri-training-CLE	**87.76**	**92.11**	**92.87**	**27.95**	1.40	10.26	$<$0.0001
tri-training-CLE (excl. pnc.)	87.54	92.61	91.68				
CONLL-X best (excl. pnc.)	84.79	90.58	89.22				
Martins et al. (excl. pnc.)	86.79	-	-				
Dutch							
mst2	80.27	84.32	84.96	23.32			
malt/mst2	81.00	84.58	85.46	24.35			
malt/mst1	80.72	84.17	85.34	26.17			
tri-training-CLE	**83.42**	**88.18**	**87.82**	**28.00**	2.42	12.74	$<$0.0001
tri-training-CLE (excl. pnc.)	81.73	86.97	86.61				
CONLL-X best (excl. pnc.)	79.19	83.57	83.89				
Martins et al. (excl. pnc.)	81.61	-	-				
German							
mst2	87.32	89.88	93.05	35.85			
malt/mst2	88.06	90.53	93.52	40.06			
malt/mst1	88.04	90.50	93.48	38.10			
tri-training-CLE	**90.41**	**93.22**	**94.61**	**43.14**	2.35	19.68	$<$0.0001
tri-training-CLE (excl. pnc.)	90.30	93.49	93.87				
CONLL-X best (excl. pnc.)	87.34	90.38	92.11				
Martins et al. (excl. pnc.)	88.66	-	-				
Portuguese							
mst2	84.83	88.44	92.04	25.69			
malt/mst2	85.39	88.80	92.59	28.13			
malt/mst1	85.00	88.39	92.23	25.69			
tri-training-CLE	**88.03**	**91.89**	**93.54**	**29.86**	2.64	18.07	$<$0.0001
tri-training-CLE (excl. pnc.)	89.18	93.69	92.43				
CONLL-X best (excl. pnc.)	87.60	91.36	91.54				
Martins et al. (excl. pnc.)	88.46	-	-				
Swedish							
mst2	81.82	87.36	87.29	27.76			
malt/mst2	84.42	89.57	88.68	31.62			
malt/mst1	84.74	89.83	89.07	31.11			
tri-training-CLE	**86.83**	**92.04**	**90.65**	**32.65**	2.09	13.70	$<$0.0001
tri-training-CLE (excl. pnc.)	86.66	92.45	89.58				
CONLL-X best (excl. pnc.)	84.58	89.50	87.39				
Martins et al. (excl. pnc.)	85.16	-	-				
AV					2.18	14.89	

Figure 4: Results on CONLL-X datasets. Scores are **including punctuation** unless otherwise noted. Δ and p-value is difference with respect to best input parser.

UAS	malt-mst2	S3VMs	self-training	orig-tri-training	co-forests	tri-training	tri-training[full]
Danish	90.50	90.47	89.68	89.66	88.79	**90.60**	92.21
Dutch	84.58	85.34	84.06	83.83	83.97	**86.07**	88.06
German	90.53	90.15	89.83	89.92	88.47	**90.81**	93.20
Portuguese	88.80	65.64	87.60	87.62	87.06	**89.16**	91.87
Swedish	89.83	81.46	89.09	89.20	88.65	**90.22**	92.24
AV	88.80	82.61	88.05	88.05	87.44	**89.37**	91.52

Figure 5: Comparison of different semi-supervised learning algorithms (10% of unlabeled data) using 2-fold CV and no reparsing, UAS **including punctuation**.

eralized tri-training.

5 Error analysis and discussion

Error reductions are higher with dependencies to the root node and long distance dependencies than with local dependencies. The table below lists the labeled attachment F_1-scores for the five datasets binned on dependency length. The average error reduction is the same for root dependencies and long distance dependencies (length >7), but significantly lower for local dependencies. This seems to indicate that large amounts of data are necessary for the parser to recover long distance dependencies.

	root	1	2	4–7	>7
Da(F_1)	98.45	96.21	92.09	88.17	90.93
– err.red	**41.34**	10.69	13.92	15.75	21.92
Du(F_1)	83.65	94.47	88.60	82.40	81.54
– err.red	28.39	16.74	20.72	17.00	**31.88**
Ge(F_1)	97.33	96.47	94.28	92.42	93.94
– err.red	26.65	19.77	17.46	25.25	**38.97**
Po(F_1)	96.23	97.05	95.17	84.80	87.11
– err.red	22.47	19.56	24.86	22.56	**26.97**
Sw(F_1)	96.37	95.67	93.46	88.42	89.57
– err.red	**32.85**	14.10	15.04	25.97	31.50
AV err.red	**30.34**	16.17	18.40	21.31	30.25

Our results for Danish, Dutch, German and Portuguese are to the best of our knowledge the best reported results in the literature. Zhang and Chan (2009) obtain a LAS of 87.20 for Swedish with transition-based parsing based on reinforcement learning. They evaluate their system on a subset of the CONLL-X datasets and obtain their (by far) best improvement on the Swedish dataset. They speculate that "the reason might be that [long distance dependencies] are not popular in Swedish". Since our parser is particularly good at long distance dependencies, this may also explain why a supervised parser outperforms our system on this dataset. Interestingly, our unlabeled attachment score is a lot better than the one reported by Zhang and Chan (2009), namely 92.45 compared to 91.84.

Generally, our UASs are better than our LASs. Since we separate attachment and labeling out in two independent steps, improvements in UAS and improvements in LA do not necessarily lead to improvements in LAS. While our average error reduction in LAS is 14.9%, our average error reductions in UAS is 23.6%. The average error reduction in LA is 14.0%. In two-stage dependency parsers or dependency parsers with joint models, improvements in UAS are typically followed by comparable improvements in LAS.

6 Conclusion

This paper showed how the stacked dependency parsers introduced in Martins et al. (2008) can be improved by inference from unlabeled data. Briefly put, we stack three diverse classifiers on triads of stacked dependency parsers and let them label unlabeled data for each other in a co-training-like architecture. Our average error reductions in LAS over the best of our stacked input parsers is 14.9%; in UAS, it is 23.6%. The code is available at http://cst.dk/anders/tridep.html.

References

Abney, Steven. 2008. *Semi-supervised learning for computational linguistics*. Chapman & Hall.

Bennett, Kristin, Ayhan Demiriz, and Richard Maclin. 2003. Exploiting unlabeled data in ensemble methods. In *KDD*.

Biemann, Chris, G. Heyer, U. Quasthoff, and M. Richter. 2007. The Leipzig corpora collection. In *Corpus Linguistics*.

Blum, Avrim and Tom Mitchell. 1998. Combining labeled and unlabeled with-co-training. In *COLT*.

Breiman, Leo. 2001. Random forests. *Machine Learning*, 45:5–32.

Buchholz, Sabine and Erwin Marsi. 2006. CONLL-X shared task on multilingual dependency parsing. In *CONLL*.

Chen, Wenliang, Yujie Zhang, and Hitoshi Isahara. 2006. Chinese chunking with tri-training learning. In *Computer processing of oriental languages*, pages 466–473. Springer, Berlin, Germany.

Didaci, Luca and Fabio Roli. 2006. Using co-training and self-training in semi-supervised multiple classifier systems. In *SSPR& SPR*, pages 522–530. Springer, Berlin, Germany.

Edmonds, J. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71:233–240.

Fishel, Mark and Joakim Nivre. 2009. Voting and stacking in data-driven dependency parsing. In *NODALIDA*.

Gimenez, Jesus and Lluis Marquez. 2004. SVMTool: a general POS tagger generator based on support vector machines. In *LREC*.

Hady, Mohamed and Friedhelm Schwenker. 2008. Co-training by committee. *International Journal of Software and Informatics*, 2:95–124.

Hall, Johan, Jens Nilsson, Joakim Nivre, Gulsen Eryigit, Beata Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? In *CONLL*.

Koehn, Philipp. 2005. Europarl: a parallel corpus for statistical machine translation. In *MT-Summit*.

Koo, Terry, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *ACL*.

Li, Ming and Zhi-Hua Zhou. 2005. Tri-training: exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541.

Li, Ming and Zhi-Hua Zhou. 2007. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE Transactions on Systems, Man and Cybernetics*, 37(6):1088–1098.

Martins, André, Dipanjan Das, Noah Smith, and Eric Xing. 2008. Stacking dependency parsers. In *EMNLP*.

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

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *HLT-EMNLP*.

Milidiu, Ruy and Julio Duarte. 2009. Improving BAS committee performance with a semi-supervised approach. In *European Symposium on Artificial Neural Networks*.

Nguyen, Tri, Le Nguyen, and Akira Shimazu. 2008. Using semi-supervised learning for question classification. *Journal of Natural Language Processing*, 15:3–21.

Nivre, Joakim and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *ACL-HLT*.

Nivre, Joakim, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser. *Natural Language Engineering*, 13(2):95–135.

Sagae, Kenji and Andrew Gordon. 2009. Clustering words by syntactic similarity improves dependency parsing of predicate-argument structures. In *IWPT*.

Sagae, Kenji and Alon Lavie. 2006. Parser combination by reparsing. In *HLT-NAACL*.

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

Sindhwani, Vikas and Sathiya Keerthi. 2006. Large scale semi-supervised linear SVMs. In *ACM SIGIR*.

Søgaard, Anders. 2010. Simple semi-supervised training of part-of-speech taggers. In *ACL*.

Spreyer, Kathrin and Jonas Kuhn. 2009. Data-driven dependency parsing of new languages using incomplete and noisy training data. In *CONLL*.

Surdeanu, Mihai and Christopher Manning. 2010. Ensemble models for dependency parsing: cheap and good? In *NAACL*.

Suzuki, Jun, Hideki Isozaki, Xavier Carreras, and Michael Collins. 2009. Semi-supervised convex training for dependency parsing. In *EMNLP*.

Wang, Qin, Dekang Lin, and Dale Schuurmans. 2008. Semi-supervised convex training for dependency parsing. In *ACL*.

Wolpert, David. 1992. Stacked generalization. *Neural Networks*, 5:241–259.

Zeman, Daniel and Zdeněk Žabokrtský. 2005. Improving parsing accuracy by combining diverse dependency parsers. In *IWPT*.

Zhang, Lidan and Kwok Chan. 2009. Dependency parsing with energy-based reinforcement learning. In *IWPT*.

Zhou, Zhi-Hua. 2009. When semi-supervised learning meets ensemble learning. In *MCS*.