Stacking Dependency Parsers

André F. T. Martins[*†]   **Dipanjan Das**[*]   **Noah A. Smith**[*]   **Eric P. Xing**[*]

[*]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[†]Instituto de Telecomunicações, Instituto Superior Técnico, Lisboa, Portugal
{afm,dipanjan,nasmith,epxing}@cs.cmu.edu

Abstract

We explore a stacked framework for learning to predict dependency structures for natural language sentences. A typical approach in graph-based dependency parsing has been to assume a factorized model, where local features are used but a global function is optimized (McDonald et al., 2005b). Recently Nivre and McDonald (2008) used the output of one dependency parser to provide features for another. We show that this is an example of *stacked learning*, in which a second predictor is trained to improve the performance of the first. Further, we argue that this technique is a novel way of approximating rich *non-local features* in the second parser, without sacrificing efficient, model-optimal prediction. Experiments on twelve languages show that stacking transition-based and graph-based parsers improves performance over existing state-of-the-art dependency parsers.

1 Introduction

In this paper we address a representation-efficiency tradeoff in statistical natural language processing through the use of **stacked learning** (Wolpert, 1992). This tradeoff is exemplified in **dependency parsing**, illustrated in Fig. 1, on which we focus in this paper:

- Exact algorithms for dependency parsing (Eisner and Satta, 1999; McDonald et al., 2005b) are tractable only when the model makes very strong, linguistically unsupportable independence assumptions, such as "arc factorization" for non-projective dependency parsing (McDonald and Satta, 2007).

- Feature-rich parsers must resort to search or greediness, (Ratnaparkhi et al., 1994; Sagae and Lavie, 2005; Hall et al., 2006), so that parsing solutions are inexact and learned models may be subject to certain kinds of bias (Lafferty et al., 2001).

A solution that leverages the complementary strengths of these two approaches—described in detail by McDonald and Nivre (2007)—was recently and successfully explored by Nivre and McDonald (2008). Our contribution begins by reinterpreting and generalizing their parser combination scheme as a **stacking of parsers**.

We give a new theoretical motivation for stacking parsers, in terms of extending a parsing model's feature space. Specifically, we view stacked learning as a way of approximating non-local features in a linear model, rather than making empirically dubious independence (McDonald et al., 2005b) or structural assumptions (e.g., projectivity, Eisner, 1996), using search approximations (Sagae and Lavie, 2005; Hall et al., 2006; McDonald and Pereira, 2006), solving a (generally NP-hard) integer linear program (Riedel and Clarke, 2006), or adding latent variables (Titov and Henderson, 2007). Notably, we introduce the use of very rich non-local *approximate* features in one parser, through the output of another parser. Related approaches are the belief propagation algorithm of Smith and Eisner (2008), and the "trading of structure for features" explored by Liang et al.
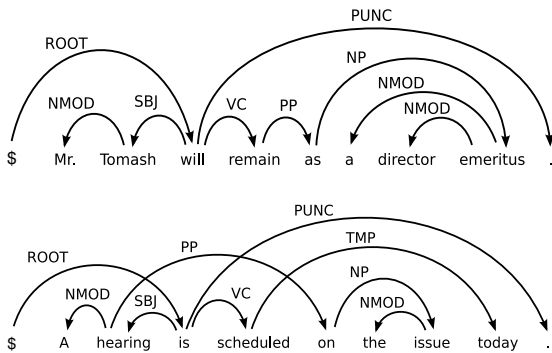
Figure 1: A projective dependency parse (top), and a non-projective dependency parse (bottom) for two English sentences; examples from McDonald and Satta (2007).

(2008).

This paper focuses on dependency parsing, which has become widely used in relation extraction (Culotta and Sorensen, 2004), machine translation (Ding and Palmer, 2005), question answering (Wang et al., 2007), and many other NLP applications. We show that stacking methods outperform the approximate "second-order" parser of McDonald and Pereira (2006) on twelve languages and can be used within that approximation to achieve even better results. These results are similar in spirit to (Nivre and McDonald, 2008), but with the following novel contributions:

- a stacking interpretation,
- a richer feature set that includes non-local features (shown here to improve performance), and
- a variety of stacking architectures.

Using stacking with rich features, we obtain results competitive with Nivre and McDonald (2008) while preserving the fast quadratic parsing time of arc-factored spanning tree algorithms.

The paper is organized as follows. We discuss related prior work on dependency parsing and stacking in §2. Our model is given in §3. A novel analysis of stacking in linear models is given in §4. Experiments are presented in §5.

2 Background and Related Work

We briefly review work on the NLP task of **dependency parsing** and the machine learning framework known as **stacked learning**.

2.1 Dependency Parsing

Dependency syntax is a lightweight syntactic representation that models a sentence as a graph where the words are vertices and syntactic relationships are directed edges (arcs) connecting heads to their arguments and modifiers.

Dependency parsing is often viewed computationally as a structured prediction problem: for each input sentence x, with n words, exponentially many candidate dependency trees $y \in \mathcal{Y}(x)$ are possible in principle. We denote each tree by its set of vertices and directed arcs, $y = (V_y, A_y)$. A legal dependency tree has $n + 1$ vertices, each corresponding to one word plus a "wall" symbol, $, assumed to be the hidden root of the sentence. In a valid dependency tree, each vertex except the root has exactly one parent. In the *projective* case, arcs cannot cross when depicted on one side of the sentence; in the *non-projective* case, this constraint is not imposed (see Fig. 1).

2.1.1 Graph-based *vs.* transition-based models

Most recent work on dependency parsing can be categorized as *graph-based* or *transition-based*. In graph-based parsing, dependency trees are scored by factoring the tree into its arcs, and parsing is performed by searching for the highest scoring tree (Eisner, 1996; McDonald et al., 2005b). Transition-based parsers model the sequence of decisions of a shift-reduce parser, given previous decisions and current state, and parsing is performed by greedily choosing the highest scoring transition out of each successive parsing state or by searching for the best sequence of transitions (Ratnaparkhi et al., 1994; Yamada and Matsumoto, 2003; Nivre et al., 2004; Sagae and Lavie, 2005; Hall et al., 2006).

Both approaches most commonly use linear models to assign scores to arcs or decisions, so that a score is a dot-product of a feature vector **f** and a learned weight vector **w**.

In sum, these two lines of research use different approximations to achieve tractability. Transition-based approaches solve a sequence of *local* problems in sequence, sacrificing global optimality guarantees and possibly expressive power (Abney et al., 1999). Graph-based methods perform *global* inference using score factorizations that correspond to strong independence assumptions (discussed in

§2.1.2). Recently, Nivre and McDonald (2008) proposed *combining* a graph-based and a transition-based parser and have shown a significant improvement for several languages by letting one of the parsers "guide" the other. Our stacked formalism (to be described in §3) generalizes this approach.

2.1.2 Arc factorization

In the successful graph-based method of McDonald et al. (2005b), an *arc factorization* independence assumption is used to ensure tractability. This assumption forbids any feature that depends on two or more arcs, permitting only "arc-factored" features (i.e. features that depend only on a single candidate arc $a \in A_y$ and on the input sequence x). This induces a decomposition of the feature vector $\mathbf{f}(x, y)$ as:

$$\mathbf{f}(x, y) = \sum_{a \in A_y} \mathbf{f}_a(x).$$

Parsing amounts to solving $\arg\max_{y \in \mathcal{Y}(x)} \mathbf{w}^\top \mathbf{f}(x, y)$, where $\mathbf{w}$ is a weight vector. With a projectivity constraint and arc factorization, the parsing problem can be solved in cubic time by dynamic programming (Eisner, 1996), and with a weaker "tree" constraint (permitting nonprojective parses) and arc factorization, a quadratic-time algorithm exists (Chu and Liu, 1965; Edmonds, 1967), as shown by McDonald et al. (2005b). In the projective case, the arc-factored assumption can be weakened in certain ways while maintaining polynomial parser runtime (Eisner and Satta, 1999), but not in the nonprojective case (McDonald and Satta, 2007), where finding the highest-scoring tree becomes NP-hard.

McDonald and Pereira (2006) adopted an approximation based on $O(n^3)$ projective parsing followed by rearrangement to permit crossing arcs, achieving higher performance. In §3 we adopt a framework that maintains $O(n^2)$ runtime (still exploiting the Chu-Liu-Edmonds algorithm) while approximating non arc-factored features.

2.2 Stacked Learning

Stacked generalization was first proposed by Wolpert (1992) and Breiman (1996) for regression. The idea is to include two "levels" of predictors. The first level, "level 0," includes one or more predictors $g_1, \dots, g_K : \mathbb{R}^d \to \mathbb{R}$; each receives input $\mathbf{x} \in \mathbb{R}^d$

and outputs a prediction $g_k(\mathbf{x})$. The second level, "level 1," consists of a single function $h : \mathbb{R}^{d+K} \to \mathbb{R}$ that takes as input $\langle \mathbf{x}, g_1(\mathbf{x}), \dots g_K(\mathbf{x}) \rangle$ and outputs a final prediction $\hat{y} = h(\mathbf{x}, g_1(\mathbf{x}), \dots g_K(\mathbf{x}))$. The predictor, then, combines an ensemble (the g_k) with a meta-predictor (h).

Training is done as follows: the training data are split into L partitions, and L instances of the level 0 predictor are trained in a "leave-one-out" basis. Then, an augmented dataset is formed by letting each instance output predictions for the partition that was left out. Finally, each level 0 predictor is trained using the original dataset, and the level 1 predictor is trained on the augmented dataset, simulating the test-time setting when h is applied to a new instance $\mathbf{x}$ concatenated with $\langle g_k(\mathbf{x}) \rangle_k$.

This framework has also been applied to classification, for example with structured data. Some applications (including here) use only one classifier at level 0; recent work includes sequence labeling (Cohen and de Carvalho, 2005) and inference in conditional random fields (Kou and Cohen, 2007). Stacking is also intuitively related to transformation-based learning (Brill, 1993).

3 Stacked Dependency Parsing

We next describe how to use stacked learning for efficient, rich-featured dependency parsing.

3.1 Architecture

The architecture consists of two levels. At level 0 we include a single dependency parser. At runtime, this "level 0 parser" g processes an input sentence x and outputs the set of predicted edges that make up its estimation of the dependency tree, $\hat{y}_0 = g(x)$. At level 1, we apply a dependency parser—in this work, always a graph-based dependency parser—that uses basic factored features *plus* new ones from the edges *predicted* by the level 0 parser. The final parser predicts parse trees as $h(x, g(x))$, so that the total runtime is additive in calculating $h(\cdot)$ and $g(\cdot)$.

The stacking framework is agnostic about the form of g and h and the methods used to learn them from data. In this work we use two well-known, publicly available dependency parsers, MSTParser (McDonald et al., 2005b),[1] which implements ex-

---

[1] http://sourceforge.net/projects/mstparser

act first-order arc-factored nonprojective parsing (§2.1.2) and approximate second-order nonprojective parsing, and MaltParser (Nivre et al., 2006), which is a state-of-the-art transition-based parser.[2] We do not alter the training algorithms used in prior work for learning these two parsers from data. Using the existing parsers as starting points, we will combine them in a variety of ways.

3.2 Training

Regardless of our choices for the specific parsers and learning algorithms at level 0 and level 1, training is done as sketched in §2.2. Let $\mathcal{D}$ be a set of training examples $\{\langle x_i, y_i \rangle\}_i$.

1. Split training data $\mathcal{D}$ into L partitions $\mathcal{D}^1, \ldots, \mathcal{D}^L$.
2. Train L instances of the level 0 parser in the following way: the l-th instance, g^l, is trained on $\mathcal{D}^{-l} = \mathcal{D} \setminus \mathcal{D}^l$. Then use g^l to output predictions for the (unseen) partition $\mathcal{D}^l$. At the end, an augmented dataset $\tilde{\mathcal{D}} = \bigcup_{l=1}^L \tilde{\mathcal{D}}^l$ is built, so that $\tilde{\mathcal{D}} = \{\langle x_i, g(x_i), y_i \rangle\}_i$.
3. Train the level 0 parser g on the original training data $\mathcal{D}$.
4. Train the level 1 parser h on the augmented training data $\tilde{\mathcal{D}}$.

The runtime of this algorithm is $O(LT_0 + T_1)$, where T_0 and T_1 are the individual runtimes required for training level 0 and level 1 alone, respectively.

4 Two Views of Stacked Parsing

We next describe two motivations for stacking parsers: as a way of augmenting the features of a graph-based dependency parser or as a way to approximate higher-order models.

4.1 Adding Input Features

Suppose that the level 1 classifier is an arc-factored graph-based parser. The feature vectors will take the form[3]

$$\mathbf{f}(x, y) = \mathbf{f}_1(x, y) \smile \mathbf{f}_2(x, \hat{y}_0, y)$$
$$= \sum_{a \in A_y} \mathbf{f}_{1,a}(x) \smile \mathbf{f}_{2,a}(x, g(x)),$$

---

[2]http://w3.msi.vxu.se/~jha/maltparser
[3]We use $\smile$ to denote vector concatenation.

where $\mathbf{f}_1(x, y) = \sum_{a \in A_y} \mathbf{f}_{1,a}(x)$ are regular arc-factored features, and $\mathbf{f}_2(x, \hat{y}_0, y) = \sum_{a \in A_y} \mathbf{f}_{2,a}(x, g(x))$ are the *stacked features*. An example of a stacked feature is a binary feature $f_{2,a}(x, g(x))$ that fires if and only if the arc a was predicted by g, i.e., if $a \in A_{g(x)}$; such a feature was used by Nivre and McDonald (2008).

It is difficult in general to decide whether the inclusion of such a feature yields a better parser, since features strongly correlate with each other. However, a popular heuristic for feature selection consists of measuring the *information gain* provided by each individual feature. In this case, we may obtain a closed-form expression for the information gain that $f_{2,a}(x, g(x))$ provides about the existence or not of the arc a in the actual dependency tree y. Let A and A' be binary random variables associated with the events $a \in A_y$ and $a' \in A_{g(x)}$, respectively. We have:

$$I(A; A') = \sum_{a,a' \in \{0,1\}} p(a, a') \log_2 \frac{p(a, a')}{p(a)p(a')}$$
$$= H(A') - \sum_{a \in \{0,1\}} p(a) H(A'|A = a).$$

Assuming, for simplicity, that at level 0 the probability of false positives equals the probability of false negatives (i.e., $P_{\text{err}} \triangleq p(a' = 0|a = 1) = p(a' = 1|a = 0)$), and that the probability of true positives equals the probability of true negatives ($1 - P_{\text{err}} = p(a' = 0|a = 0) = p(a' = 1|a = 1)$), the expression above reduces to:

$$I(A; A') = H(A') + P_{\text{err}} \log_2 P_{\text{err}}$$
$$+ (1 - P_{\text{err}}) \log_2(1 - P_{\text{err}})$$
$$= H(A') - H_{\text{err}},$$

where H_{err} denotes the entropy of the probability of error on each arc's prediction by the level 0 classifier. If $P_{\text{err}} \leq 0.5$ (i.e. if the level 0 classifier is better than random), then the information gain provided by this simple stacked feature increases with (a) the accuracy of the level 0 classifier, and (b) the entropy $H(A')$ of the distribution associated with its arc predictions.

4.2 Approximating Non-factored Features

Another way of interpreting the stacking framework is as a means to approximate a higher order model,

such as one that is not arc-factored, by using stacked features that make use of the predicted structure *around* a candidate arc. Consider a second-order model where the features decompose by arc and by arc *pair*:

$$\mathbf{f}(x,y) = \sum_{a_1 \in A_y} \left(\mathbf{f}_{a_1}(x) \smile \sum_{a_2 \in A_y} \mathbf{f}_{a_1,a_2}(x) \right).$$

Exact parsing under such model, with arbitrary second-order features, is intractable (McDonald and Satta, 2007). Let us now consider a stacked model in which the level 0 predictor outputs a parse $\hat{y}$. At level 1, we use arc-factored features that may be written as

$$\tilde{\mathbf{f}}(x,y) = \sum_{a_1 \in A_y} \left(\mathbf{f}_{a_1}(x) \smile \sum_{a_2 \in A_{\hat{y}}} \mathbf{f}_{a_1,a_2}(x) \right);$$

this model differs from the previous one only by replacing A_y by $A_{\hat{y}}$ in the index set of the second summation. Since $\hat{y}$ is given, this makes the latter model arc-factored, and therefore, tractable. We can now view $\tilde{\mathbf{f}}(x,y)$ as an approximation of $\mathbf{f}(x,y)$; indeed, we can bound the *score approximation error*,

$$\Delta s(x,y) = \left| \tilde{\mathbf{w}}^\top \tilde{\mathbf{f}}(x,y) - \mathbf{w}^\top \mathbf{f}(x,y) \right|,$$

where $\tilde{\mathbf{w}}$ and $\mathbf{w}$ stand respectively for the parameters learned for the stacked model and those that would be learned for the (intractable) exact second order model. We can bound $\Delta s(x,y)$ by splitting it into two terms: $\Delta s(x,y) =$

$$\left| (\tilde{\mathbf{w}} - \mathbf{w})^\top \tilde{\mathbf{f}}(x,y) + \mathbf{w}^\top (\tilde{\mathbf{f}}(x,y) - \mathbf{f}(x,y)) \right|$$
$$\leq \underbrace{\left| (\tilde{\mathbf{w}} - \mathbf{w})^\top \tilde{\mathbf{f}}(x,y) \right|}_{\triangleq \Delta s_{\mathrm{tr}}(x,y)} + \underbrace{\left| \mathbf{w}^\top (\tilde{\mathbf{f}}(x,y) - \mathbf{f}(x,y)) \right|}_{\triangleq \Delta s_{\mathrm{dec}}(x,y)};$$

where we introduced the terms Δs_{tr} and Δs_{dec} that reflect the portion of the score approximation error that are due to *training error* (i.e., different parameterizations of the exact and approximate models) and *decoding error* (same parameterizations, but different feature vectors). Using Hölder's inequality, the former term can be bounded as:

$$\begin{aligned} \Delta s_{\mathrm{tr}}(x,y) &= \left| (\tilde{\mathbf{w}} - \mathbf{w})^\top \tilde{\mathbf{f}}(x,y) \right| \\ &\leq \|\tilde{\mathbf{w}} - \mathbf{w}\|_1 \cdot \|\tilde{\mathbf{f}}(x,y)\|_\infty \\ &\leq \|\tilde{\mathbf{w}} - \mathbf{w}\|_1; \end{aligned}$$

where $\|.\|_1$ and $\|.\|_\infty$ denote the ℓ_1-norm and sup-norm, respectively, and the last inequality holds when the features are binary (so that $\|\tilde{\mathbf{f}}(x,y)\|_\infty \leq 1$). The proper way to bound the term $\|\tilde{\mathbf{w}} - \mathbf{w}\|_1$ depends on the training algorithm. As for the decoding error term, it can bounded for a given weight vector $\mathbf{w}$, sentence x, candidate tree y, and level 0 prediction $\hat{y}$. Decomposing the weighted vector as $\mathbf{w} = \mathbf{w}_1 \smile \mathbf{w}_2$, $\mathbf{w}_2$ being the sub-vector associated with the second-order features, we have respectively: $\Delta s_{\mathrm{dec}}(x,y) =$

$$\left| \mathbf{w}^\top (\tilde{\mathbf{f}}(x,y) - \mathbf{f}(x,y)) \right|$$
$$= \left| \sum_{a_1 \in A_y} \mathbf{w}_2^\top \left(\sum_{a_2 \in A_{\hat{y}}} \mathbf{f}_{a_1,a_2}(x) - \sum_{a_2 \in A_y} \mathbf{f}_{a_1,a_2}(x) \right) \right|$$
$$\leq \sum_{a_1 \in A_y} \sum_{a_2 \in A_{\hat{y}} \Delta A_y} \left| \mathbf{w}_2^\top \mathbf{f}_{a_1,a_2}(x) \right|$$
$$\leq \sum_{a_1 \in A_y} |A_{\hat{y}} \Delta A_y| \cdot \max_{a_2 \in A_{\hat{y}} \Delta A_y} \left| \mathbf{w}_2^\top \mathbf{f}_{a_1,a_2}(x) \right|$$
$$= \sum_{a_1 \in A_y} 2L(y,\hat{y}) \cdot \max_{a_2 \in A_{\hat{y}} \Delta A_y} \left| \mathbf{w}_2^\top \mathbf{f}_{a_1,a_2}(x) \right|,$$

where $A_{\hat{y}} \Delta A_y \triangleq (A_{\hat{y}} - A_y) \cup (A_y - A_{\hat{y}})$ denotes the *symmetric difference* of the sets $A_{\hat{y}}$ and A_y, which has cardinality $2L(y,\hat{y})$, *i.e.*, twice the Hamming distance between the sequences of heads that characterize y and the predicted parse $\hat{y}$. Using Hölder's inequality, we have both

$$\left| \mathbf{w}_2^\top \mathbf{f}_{a_1,a_2}(x) \right| \leq \|\mathbf{w}_2\|_1 \cdot \|\mathbf{f}_{a_1,a_2}(x)\|_\infty$$
$$\text{and } \left| \mathbf{w}_2^\top \mathbf{f}_{a_1,a_2}(x) \right| \leq \|\mathbf{w}_2\|_\infty \cdot \|\mathbf{f}_{a_1,a_2}(x)\|_1.$$

Assuming that all features are binary valued, we have that $\|\mathbf{f}_{a_1,a_2}(x)\|_\infty \leq 1$ and that $\|\mathbf{f}_{a_1,a_2}(x)\|_1 \leq N_{f,2}$, where $N_{f,2}$ denotes the maximum number of active second order features for any possible pair of arcs (a_1, a_2). Therefore:

$$\Delta s_{\mathrm{dec}}(x,y) \leq 2nL(y,\hat{y}) \min\{\|\mathbf{w}_2\|_1, N_{f,2} \cdot \|\mathbf{w}_2\|_\infty\},$$

where n is the sentence length. Although this bound can be loose, it suggests (intuitively) that the score approximation degrades as the predicted tree $\hat{y}$ gets farther away from the true tree y (in Hamming distance). It also degrades with the magnitude of weights associated with the second-order features,

Name	Description
PredEdge	Indicates whether the candidate edge was present, and what was its label.
Sibling	Lemma, POS, link label, distance and direction of attachment of the previous and and next predicted siblings
GrandParents	Lemma, POS, link label, distance and direction of attachment of the grandparent of the current modifier
PredHead	Predicted head of the candidate modifier (if PredEdge=0)
AllChildren	Sequence of POS and link labels of all the predicted children of the candidate head

Table 1: Feature sets derived from the level 0 parser.

Subset	Description
A	PredEdge
B	PredEdge+Sibling
C	PredEdge+Sibling+GrandParents
D	PredEdge+Sibling+GrandParents+PredHead
E	PredEdge+Sibling+GrandParents+PredHead+ AllChildren

Table 2: Combinations of features enumerated in Table 1 used for stacking. A is a replication of (Nivre and Mc-Donald, 2008), except for the modifications described in footnote 4.

which suggests that a separate regularization of the first-order and stacked features might be beneficial in a stacking framework.

As a side note, if we set each component of the weight vector to one, we obtain a bound on the ℓ_1-norm of the feature vector difference, $\left\| \tilde{\mathbf{f}}(x, y) - \mathbf{f}(x, y) \right\|_1 \leq 2nL(y, \hat{y})N_{f,2}$.

5 Experiments

In the following experiments we demonstrate the effectiveness of stacking parsers. As noted in §3.1, we make use of two component parsers, the graph-based MSTParser and the transition-based MaltParser.

5.1 Implementation and Experimental Details

The publicly available version of MSTParser performs parsing and labeling jointly. We adapted this system to first perform unlabeled parsing, then label the arcs using a log-linear classifier with access to the full unlabeled parse (McDonald et al., 2005a;

McDonald et al., 2005b; McDonald and Pereira, 2006). In stacking experiments, the arc labels from the level 0 parser are also used as a feature.[4]

In the following subsections, we refer to our modification of the MSTParser as MST_{1O} (the arc-factored version) and MST_{2O} (the second-order arc-pair-factored version). All our experiments use the non-projective version of this parser. We refer to the MaltParser as $Malt$.

We report experiments on twelve languages from the CoNLL-X shared task (Buchholz and Marsi, 2006).[5] All experiments are evaluated using the *labeled attachment score* (LAS), using the default settings.[6] Statistical significance is measured using Dan Bikel's randomized parsing evaluation comparator with 10,000 iterations.[7] The additional features used in the level 1 parser are enumerated in Table 1 and their various subsets are depicted in Table 2. The PredEdge features are exactly the six features used by Nivre and McDonald (2008) in their **MST**$_{Malt}$ parser; therefore, feature set A is a replication of this parser except for modifications noted in footnote 4. In all our experiments, the number of partitions used to create $\tilde{\mathcal{D}}$ is $L = 2$.

5.2 Experiment: $MST_{2O} + MST_{2O}$

Our first experiment stacks the highly accurate MST_{2O} parser with itself. At level 0, the parser uses only the standard features (§5.1), and at level 1, these are augmented by various subsets of features of x along with the output of the level 0 parser, $g(x)$ (Table 2). The results are shown in Table 3. While we see improvements over the single-parser baseline

---

[4] We made other modifications to MSTParser, implementing many of the successes described by (McDonald et al., 2006). Our version of the code is publicly available at http://www.ark.cs.cmu.edu/MSTParserStacked. The modifications included an approximation to lemmas for datasets without lemmas (three-character prefixes), and replacing morphology/word and morphology/lemma features with morphology/POS features.

[5] The CoNLL-X shared task actually involves thirteen languages; our experiments do not include Czech (the largest dataset), due to time constraints. Therefore, the average results plotted in the last rows of Tables 3, 4, and 5 are not directly comparable with previously published averages over thirteen languages.

[6] http://nextens.uvt.nl/~conll/software.html

[7] http://www.cis.upenn.edu/~dbikel/software.html

	MST_{2O}	$+MST_{2O,A}$	$+MST_{2O,B}$	$+MST_{2O,C}$	$+MST_{2O,D}$	$+MST_{2O,E}$
Arabic	67.88	66.91	67.41	67.68	67.37	**68.02**
Bulgarian	87.31	87.39	87.03	**87.61**	87.57	87.55
Chinese	**87.57**	87.16	87.24	87.48	87.42	87.48
Danish	85.27	85.39	**85.61**	85.57	85.43	85.57
Dutch	79.99	79.79	79.79	79.83	**80.17**	80.13
German	**87.44**	86.92	87.32	87.32	87.26	87.04
Japanese	90.93	**91.41**	91.21	91.35	91.11	91.19
Portuguese	87.12	**87.26**	86.88	87.02	87.04	86.98
Slovene	74.02	**74.30**	**74.30**	74.00	74.14	73.94
Spanish	82.43	82.17	82.35	**82.81**	82.53	82.75
Swedish	82.87	82.99	82.95	82.51	**83.01**	82.69
Turkish	**60.11**	59.47	59.25	59.47	59.45	59.31
Average	**81.08**	80.93	80.94	81.05	81.04	81.05

Table 3: Results of stacking MST_{2O} with itself at both level 0 and level 1. Column 2 enumerates LAS for MST_{2O}. Columns 3–6 enumerate results for four different stacked feature subsets. Bold indicates best results for a particular language.

for nine languages, the improvements are small (less than 0.5%). One of the biggest concerns about this model is the fact that it stacks two predictors that are very similar in nature: both are graph-based and share the features $\mathbf{f}_{1,a}(x)$. It has been pointed out by Breiman (1996), among others, that the success of ensemble methods like stacked learning strongly depends on how uncorrelated the individual decisions made by each predictor are from the others' decisions.[8] This experiment provides further evidence for the claim.

5.3 Experiment: $Malt + MST_{2O}$

We next use MaltParser at level 0 and the *second-order* arc-pair-factored MST_{2O} at level 1. This extends the experiments of Nivre and McDonald (2008), replicated in our feature subset A.

Table 4 enumerates the results. Note that the best-performing stacked configuration for each and every language outperforms MST_{2O}, corroborating results reported by Nivre and McDonald (2008). The best performing stacked configuration outperforms *Malt* as well, except for Japanese and Turkish. Further, our non-arc-factored features largely outperform subset A, except on Bulgarian, Chinese,

and Japanese. On average, the best feature configuration is E, which is statistically significant over *Malt* and MST_{2O} with $p < 0.0001$, and over feature subset A with $p < 0.01$.

5.4 Experiment: $Malt + MST_{1O}$

Finally, we consider stacking MaltParser with the first-order, arc-factored MSTParser. We view this approach as perhaps the most promising, since it is an exact parsing method with the quadratic runtime complexity of MST_{1O}.

Table 5 enumerates the results. For all twelve languages, some stacked configuration outperforms MST_{1O} and also, surprisingly, MST_{2O}, the second order model. This provides empirical evidence that using rich features from MaltParser at level 0, a stacked level 1 first-order MSTParser can outperform the second-order MSTParser.[9] In only two cases (Japanese and Turkish), the MaltParser slightly outperforms the stacked parser.

On average, feature configuration D performs the best, and is statistically significant over *Malt*, MST_{1O}, and MST_{2O} with $p < 0.0001$, and over feature subset A with $p < 0.05$. Encouragingly, this configuration is barely outperformed by configura-

---

[8]This claim has a parallel in the cotraining method (Blum and Mitchell, 1998), whose performance is bounded by the degree of independence between the two feature sets.

[9]Recall that MST_{2O} uses approximate *search*, as opposed to stacking, which uses approximate *features*.

	Malt	MST_{2O}	Malt + MST_{2O},A	Malt + MST_{2O},B	Malt + MST_{2O},C	Malt + MST_{2O},D	Malt + MST_{2O},E
Arabic	66.71	67.88	68.56	**69.12**	68.64	68.34	68.92
Bulgarian	87.41	87.31	**88.99**	88.89	88.89	88.93	88.91
Chinese	86.92	87.57	**88.41**	88.31	88.29	88.13	**88.41**
Danish	84.77	85.27	86.45	86.67	**86.79**	86.13	86.71
Dutch	78.59	79.99	80.75	81.47	81.47	**81.51**	81.29
German	85.82	87.44	88.16	88.50	88.56	**88.68**	88.38
Japanese	**91.65**	90.93	91.63	91.43	91.59	91.61	91.49
Portuguese	87.60	87.12	88.00	88.24	**88.30**	88.18	88.22
Slovene	70.30	74.02	76.62	76.00	76.60	76.18	**76.72**
Spanish	81.29	82.43	83.09	**83.73**	83.47	83.21	83.43
Swedish	84.58	82.87	84.92	84.60	84.80	**85.16**	84.88
Turkish	**65.68**	60.11	64.35	64.51	64.51	65.07	65.21
Average	80.94	81.08	82.52	82.58	82.65	82.59	**82.71**

Table 4: Results of stacking $Malt$ and MST_{2O} at level 0 and level 1, respectively. Columns 2–4 enumerate LAS for $Malt$, MST_{2O} and $Malt + MST_{2O}$ as in Nivre and McDonald (2008). Columns 5–8 enumerate results for four other stacked feature configurations. Bold indicates best result for a language.

tion A of $Malt + MST_{2O}$ (see Table 4), the difference being statistically insignificant ($p > 0.05$). This shows that stacking $Malt$ with the exact, arc-factored MST_{1O} bridges the difference between the individual MST_{1O} and MST_{2O} models, by approximating higher order features, but maintaining an $O(n^2)$ runtime and finding the model-optimal parse.

5.5 Disagreement as a Confidence Measure

In pipelines or semisupervised settings, it is useful when a parser can provide a *confidence* measure alongside its predicted parse tree. Because stacked predictors use ensembles with observable outputs, differences among those outputs may be used to estimate confidence in the final output. In stacked dependency parsing, this can be done (for example) by measuring the Hamming distance between the outputs of the level 0 and 1 parsers, $L(g(x), h(x))$. Indeed, the bound derived in §4.2 suggests that the second-order approximation degrades for candidate parses y that are Hamming-far from $g(x)$; therefore, if $L(g(x), h(x))$ is large, the best score $s(x, h(x))$ may well be "biased" due to misleading neighboring information provided by the level 0 parser.

We illustrate this point with an empirical analysis of the level 0/1 disagreement for the set of experiments described in §5.3; namely, we compare the
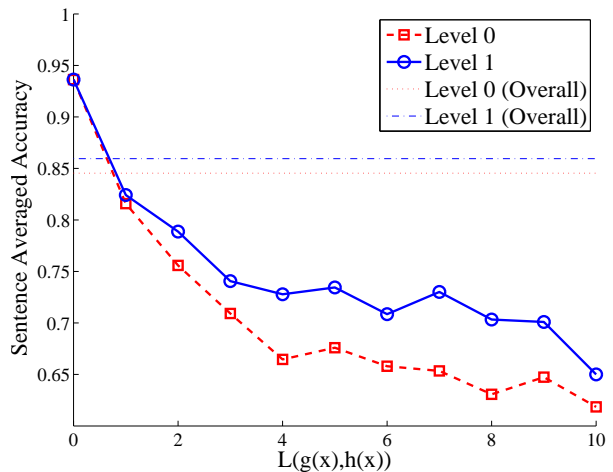

Figure 2: Accuracy as a function of token disagreement between level 0 and level 1. The x-axis is the Hamming distance $L(g(x), h(x))$, i.e., the number of tokens where level 0 and level 1 disagree. The y-axis is the accuracy averaged over sentences that have the specified Hamming distance, both for level 0 and level 1.

	$Malt$	MST_{1O}	MST_{2O}	$Malt + MST_{1O,A}$	$Malt + MST_{1O,B}$	$Malt + MST_{1O,C}$	$Malt + MST_{1O,D}$	$Malt + MST_{1O,E}$
Arabic	66.71	66.81	67.88	68.40	68.50	68.20	68.42	**68.68**
Bulgarian	87.41	86.65	87.31	88.55	88.67	88.75	88.71	**88.79**
Chinese	86.92	86.60	87.57	87.67	87.73	**87.83**	87.67	87.61
Danish	84.77	84.87	85.27	**86.59**	86.27	86.21	86.35	86.15
Dutch	78.59	78.95	79.99	80.53	81.51	80.71	**81.61**	81.37
German	85.82	86.26	87.44	88.18	88.30	88.20	88.36	**88.42**
Japanese	**91.65**	91.01	90.93	91.55	91.53	91.51	91.43	91.57
Portuguese	87.60	86.28	87.12	88.16	88.26	**88.46**	88.26	88.36
Slovene	70.30	73.96	74.02	75.84	75.64	75.42	**75.96**	75.64
Spanish	81.29	81.07	82.43	82.61	**83.13**	**83.13**	83.09	82.99
Swedish	84.58	81.88	82.87	**84.86**	84.62	84.64	84.82	84.76
Turkish	**65.68**	59.63	60.11	64.49	64.97	64.47	64.63	64.61
Average	80.94	80.33	81.08	82.28	82.42	82.29	**82.44**	82.41

Table 5: Results of stacking *Malt* and MST_{1O} at level 0 and level 1, respectively. Columns 2–4 enumerate LAS for *Malt*, MST_{1O} and MST_{2O}. Columns 5–9 enumerate results for five different stacked feature configurations. Bold indicates the best result for a language.

level 0 and level 1 predictions under the best overall configuration (configuration E of $Malt + MST_{2O}$). Figure 2 depicts accuracy as a function of level 0-level 1 disagreement (in number of tokens), averaged over all datasets.

We can see that performance degrades steeply when the disagreement between levels 0 and 1 increases in the range 0–4, and then behaves more irregularly but keeping the same trend. This suggests that the Hamming distance $L(g(x), h(x))$ is informative about parser performance and may be used as a confidence measure.

6 Conclusion

In this work, we made use of stacked learning to improve dependency parsing. We considered an architecture with two layers, where the output of a standard parser in the first level provides new features for a parser in the subsequent level. During learning, the second parser learns to correct mistakes made by the first one. The novelty of our approach is in the exploitation of higher-order predicted edges to simulate non-local features in the second parser. We provided a novel interpretation of stacking as feature approximation, and our experimental results show rich-featured stacked parsers outperforming state-of-the-art single-layer and ensemble parsers. No-

tably, using a simple arc-factored parser at level 1, we obtain an exact $O(n^2)$ stacked parser that outperforms earlier approximate methods (McDonald and Pereira, 2006).

Acknowledgments

References

S. P. Abney, D. A. McAllester, and F. Pereira. 1999. Relating probabilistic grammars and automata. In *Proceedings of ACL*.

A. Blum and T. Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of COLT*.

L. Breiman. 1996. Stacked regressions. *Machine Learning*, 24:49.

E. Brill. 1993. *A Corpus-Based Approach to Language Learning*. Ph.D. thesis, University of Pennsylvania.

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

Y. J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

W. W. Cohen and V. Rocha de Carvalho. 2005. Stacked sequential learning. In *Proceedings of IJCAI*.

A. Culotta and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of ACL*.

Y. Ding and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammar. In *Proceedings of ACL*.

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

J. Eisner and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of ACL*.

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

J. Hall, J. Nivre, and J. Nilsson. 2006. Discriminative classifiers for deterministic dependency parsing. In *Proceedings of ACL*.

Z. Kou and W. W. Cohen. 2007. Stacked graphical models for efficient inference in Markov random fields. In *Proceedings of SDM*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*.

P. Liang, H. Daumé, and D. Klein. 2008. Structure compilation: trading structure for features. In *Proceedings of ICML*.

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

R. T. McDonald and F. C. N. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*.

R. McDonald and G. Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of IWPT*.

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

R. T. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT-EMNLP*.

R. McDonald, K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings CoNLL*.

J. Nivre and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-HLT*.

J. Nivre, J. Hall, and J. Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of CoNLL*.

J. Nivre, J. Hall, J. Nilsson, G. Eryiğit, and S. Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of CoNLL*.

A. Ratnaparkhi, S. Roukos, and R. T. Ward. 1994. A maximum entropy model for parsing. In *Proceedings of ICSLP*.

S. Riedel and J. Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of EMNLP*.

K. Sagae and A. Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of IWPT*.

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

I. Titov and J. Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of IWPT*.

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

D. Wolpert. 1992. Stacked generalization. *Neural Networks*, 5(2):241–260.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*.