Sentence Simplification for Semantic Role Labeling

David Vickrey and **Daphne Koller**
Stanford University
Stanford, CA 94305-9010
{dvickrey,koller}@cs.stanford.edu

Abstract

Parse-tree paths are commonly used to incorporate information from syntactic parses into NLP systems. These systems typically treat the paths as atomic (or nearly atomic) features; these features are quite sparse due to the immense variety of syntactic expression. In this paper, we propose a general method for learning how to iteratively simplify a sentence, thus decomposing complicated syntax into small, easy-to-process pieces. Our method applies a series of hand-written transformation rules corresponding to basic syntactic patterns — for example, one rule "depassivizes" a sentence. The model is parameterized by *learned* weights specifying preferences for some rules over others. After applying all possible transformations to a sentence, we are left with a set of candidate simplified sentences. We apply our simplification system to semantic role labeling (SRL). As we do not have labeled examples of correct simplifications, we use labeled training data for the SRL task to jointly learn both the weights of the simplification model and of an SRL model, treating the simplification as a hidden variable. By extracting and labeling simplified sentences, this combined simplification/SRL system better generalizes across syntactic variation. It achieves a statistically significant 1.2% F1 measure increase over a strong baseline on the Conll-2005 SRL task, attaining near-state-of-the-art performance.

1 Introduction

In *semantic role labeling* (SRL), given a sentence containing a *target verb*, we want to label the semantic arguments, or roles, of that verb. For the verb "eat", a correct labeling of "Tom ate a salad" is {ARG0(Eater)="Tom", ARG1(Food)="salad"}.

Current semantic role labeling systems rely primarily on syntactic features in order to identify and classify roles. Features derived from a syntactic parse of the sentence have proven particularly useful (Gildea & Jurafsky, 2002). For example, the syntactic subject of "give" is nearly always the Giver. Path features allow systems to capture both general patterns, e.g., that the ARG0 of a sentence tends to be the subject of the sentence, and specific usage, e.g., that the ARG2 of "give" is often a post-verbal prepositional phrase headed by "to". An example sentence with extracted path features is shown in Figure 1.

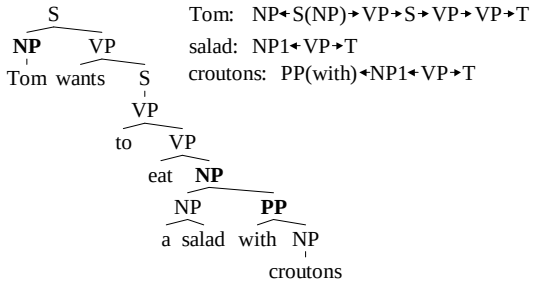

Figure 1: Parse with path features for verb "eat".

A major problem with this approach is that the path from an argument to the verb can be quite complicated. In the sentence "He expected to receive a prize for winning," the path from "win" to its ARG0, "he", involves the verbs "expect" and "receive" and the preposition "for." The corresponding path through the parse tree likely occurs a relatively small number of times (or not at all) in the training corpus. If the test set contained exactly the same sentence but with "expected" replaced by "did not expect" we would extract a different parse path feature; therefore, as far as the classifier is concerned, the syntax of the two sentences is totally unrelated.

In this paper we learn a mapping from full, complicated sentences to simplified sentences. For example, given a correct parse, our system simplifies the above sentence with target verb "win" to "He won." Our method combines hand-written syntactic simplification rules with machine learning, which

Proceedings of ACL-08: HLT, pages 344–352,
Columbus, Ohio, USA, June 2008.

determines which rules to prefer. We then use the output of the simplification system as input to a SRL system that is trained to label simplified sentences.

Compared to previous SRL models, our model has several qualitative advantages. First, we believe that the simplification process, which represents the syntax as a set of local syntactic transformations, is more linguistically satisfying than using the entire parse path as an atomic feature. Improving the simplification process mainly involves adding more linguistic knowledge in the form of simplification rules. Second, labeling simple sentences is much easier than labeling raw sentences and allows us to generalize more effectively across sentences with differing syntax. This is particularly important for verbs with few labeled training instances; using training examples as efficiently as possible can lead to considerable gains in performance. Third, our model is very effective at sharing information across verbs, since most of our simplification rules apply equally well regardless of the target verb.

A major difficulty in learning to simplify sentences is that we do not have labeled data for this task. To address this problem, we simultaneously train our simplification system and the SRL system. We treat the correct simplification as a hidden variable, using labeled SRL data to guide us towards "more useful" simplifications. Specifically, we train our model discriminatively to predict the correct role labeling assignment given an input sentence, treating the simplification as a hidden variable.

Applying our combined simplification/SRL model to the Conll 2005 task, we show a significant improvement over a strong baseline model. Our model does best on verbs with little training data and on instances with paths that are rare or have never been seen before, matching our intuitions about the strengths of the model. Our model outperforms all but the best few Conll 2005 systems, each of which uses multiple different automatically-generated parses (which would likely improve our model).

2 Sentence Simplification

We will begin with an example before describing our model in detail. Figure 2 shows a series of transformations applied to the sentence "I was not given a chance to eat," along with the interpretation of each transformation. Here, the target verb is "eat."

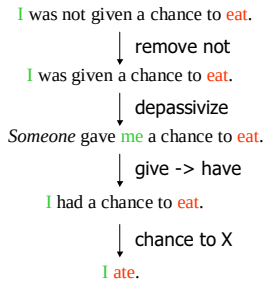

Figure 2: Example simplification

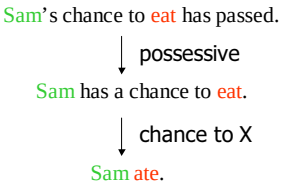

Figure 3: Shared simplification structure

There are several important things to note. First, many of the steps do lose some semantic information; clearly, having a chance to eat is not the same as eating. However, since we are interested only in labeling the core arguments of the verb (which in this case is simply the Eater, "I"), it is not important to maintain this information. Second, there is more than one way to choose a set of rules which lead to the desired final sentence "I ate." For example, we could have chosen to include a rule which went directly from the second step to the fourth. In general, the rules were designed to allow as much reuse of rules as possible. Figure 3 shows the simplification of "Sam's chance to eat has passed" (again with target verb "eat"); by simplifying both of these sentences as "X had a chance to Y", we are able to use the same final rule in both cases.

Of course, there may be more than one way to simplify a sentence for a given rule set; this ambiguity is handled by learning which rules to prefer.

In this paper, we use simplification to mean something which is closer to *canonicalization* that *summarization*. Thus, given an input sentence, our goal is not to produce a single shortened sentence which contains as much information from the original sentence as possible. Rather, the goal is, for *each* verb in the sentence, to produce a "simple" sentence which is in a particular canonical form (described below) relative to that verb.

3 Transformation Rules

A *transformation rule* takes as input a parse tree and produces as output a different, changed parse tree. Since our goal is to produce a simplified version of the sentence, the rules are designed to bring all sentences toward the same common format.

A rule (see left side of Figure 4) consists of two

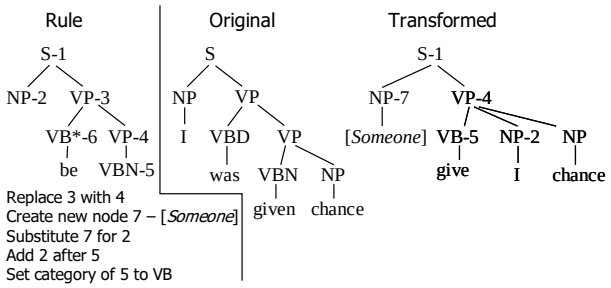

Figure 4: Rule for depassivizing a sentence

parts. The first is a "tree regular expression" which is most simply viewed as a tree fragment with optional constraints at each node. The rule assigns numbers to each node which are referred to in the second part of the rule. Formally, a rule node X *matches* a parse-tree node A if: (1) All constraints of node X (e.g., constituent category, head word, etc.) are satisfied by node A. (2) For each child node Y of X, there is a child B of A that matches Y; two children of X cannot be matched to the same child B. There are no other requirements. A can have other children besides those matched, and leaves of the rule pattern can match to internal nodes of the parse (corresponding to entire phrases in the original sentence). For example, the same rule is used to simplify both "I had a chance to eat," and "I had a chance to eat a sandwich," (into "I ate," and "I ate a sandwich,"). The insertion of the phrase "a sandwich" does not prevent the rule from matching.

The second part of the rule is a series of simple steps that are applied to the matched nodes. For example, one type of simple step applied to the pair of nodes (X,Y) removes X from its current parent and adds it as the final child of Y. Figure 4 shows the depassivizing rule and the result of applying it to the sentence "I was given a chance." The transformation steps are applied sequentially from top to bottom. Note that any nodes not matched are unaffected by the transformation; they remain where they are relative to their parents. For example, "chance" is not matched by the rule, and thus remains as a child of the VP headed by "give."

There are two significant pieces of "machinery" in our current rule set. The first is the idea of a *floating node*, used for locating an argument within a subordinate clause. For example, in the phrases "The cat that ate the mouse", "The seed that the mouse ate", and "The person we gave the gift to", the modified nouns ("cat", "seed", and "person", respectively) all should be placed in different positions in the subordinate clauses (subject, direct object, and object of "to") to produce the phrases "The cat ate the mouse," "The mouse ate the seed", and "We gave the gift to the person." We handle these phrases by placing a floating node in the subordinate clause which points to the argument; other rules try to place the floating node into each possible position in the sentence.

Rule Category	#	Original	Simplified
Sentence normalization	24	Thursday, I slept.	I slept Thursday.
Sentence extraction	4	I said he slept.	He slept.
Passive	5	I was hit by a car.	A car hit me.
Misc Collapsing/Rewriting	20	John, a lawyer, …	John is a lawyer.
Conjunctions	8	I ate and slept.	I ate.
Verb Collapsing/Rewriting	14	I must eat.	I eat.
Verb Raising/Control (basic)	17	I want to eat.	I eat.
Verb RC (ADJP/ADVP)	6	I am likely to eat.	I eat.
Verb RC (Noun)	7	I have a chance to eat.	I eat.
Modified nouns	8	The food I ate …	Float(The food) I ate.
Floating nodes	5	Float(The food) I ate.	I ate the food.
Inverted sentences	7	Nor will I eat.	I will eat.
Questions	7	Will I eat?	I will eat.
Possessive	7	John's chance to eat…	John has a chance to eat.
Verb acting as PP/NP	7	Including tax, the total…	The total includes tax.
"Make" rewrites	8	Salt makes food tasty.	Food is tasty.

Table 1: Rule categories with sample simplifications. Target verbs are underlined.

The second construct is a system for keeping track of whether a sentence has a subject, and if so, what it is. A subset of our rule set normalizes the input sentence by moving modifiers after the verb, leaving either a single phrase (the subject) or nothing before the verb. For example, the sentence "Before leaving, I ate a sandwich," is rewritten as "I ate a sandwich before leaving." In many cases, keeping track of the presence or absence of a subject greatly reduces the set of possible simplifications.

Altogether, we currently have 154 (mostly unlexicalized) rules. Our general approach was to write very conservative rules, i.e., avoid making rules with low precision, as these can quickly lead to a large blowup in the number of generated simple sentences. Table 1 shows a summary of our rule-set, grouped by type. Note that each row lists only one possible sentence and simplification rule from that

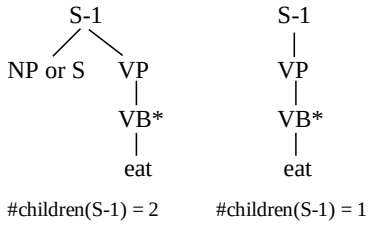

Figure 5: Simple sentence constraints for "eat"

category; many of the categories handle a variety of syntax patterns. The two examples without target verbs are helper transformations; in more complex sentences, they can enable further simplifications. Another thing to note is that we use the terms Raising/Control (RC) very loosely to mean situations where the subject of the target verb is displaced, appearing as the subject of another verb (see table).

Our rule set was developed by analyzing performance and coverage on the PropBank WSJ training set; neither the development set nor (of course) the test set were used during rule creation.

4 Simple Sentence Production

We now describe how to take a set of rules and produce a set of candidate simple sentences. At a high level, the algorithm is very simple. We maintain a set of derived parses S which is initialized to contain only the original, untransformed parse. One iteration of the algorithm consists of applying every possible matching transformation rule to every parse in S, and adding all resulting parses to S. With carefully designed rules, repeated iterations are guaranteed to converge; that is, we eventually arrive at a set $\hat{S}$ such that if we apply an iteration of rule application to $\hat{S}$, no new parses will be added. Note that we simplify the whole sentence without respect to a particular verb. Thus, this process only needs to be done once per sentence (not once per verb).

To label arguments of a particular target verb, we remove any parse from our set which does not match one of the two templates in Figure 5 (for verb "eat"). These select simple sentences that have all non-subject modifiers moved to the predicate and "eat" as the main verb. Note that the constraint VB* indicates any terminal verb category (e.g., VBN, VBD, etc.) A parse that matches one of these templates is called a *valid simple sentence*; this is exactly the canonicalized version of the sentence which our simplification rules are designed to produce.

This procedure is quite expensive; we have to copy the entire parse tree at each step, and in general, this procedure could generate an exponential number of transformed parses. The first issue can be solved, and the second alleviated, using a dynamic-programming data structure similar to the one used to store parse forests (as in a chart parser). This data structure is not essential for exposition; we delay discussion until Section 7.

5 Labeling Simple Sentences

For a particular sentence/target verb pair s, v, the output from the previous section is a set $S^{sv} = \{t_i^{sv}\}_i$ of valid simple sentences. Although labeling a simple sentence is easier than labeling the original sentence, there are still many choices to be made. There is one key assumption that greatly reduces the search space: in a simple sentence, only the subject (if present) and direct modifiers of the target verb can be arguments of that verb.

On the training set, we now extract a set of *role patterns* $G^v = \{g_j^v\}_j$ for each verb v. For example, a common role pattern for "give" is that of "I gave him a sandwich". We represent this pattern as $g_1^{give} = \{ARG0 = Subject\ NP,\ ARG1 = Postverb\ NP2,\ ARG2 = Postverb\ NP1\}$. Note that this is one atomic pattern; thus, we are keeping track not just of occurrences of particular roles in particular places in the simple sentence, but also how those roles co-occur with other roles.

For a particular simple sentence t_i^{sv}, we apply all extracted role patterns g_j^v to t_i^{sv}, obtaining a set of possible role labelings. We call a simple sentence/role labeling pair a *simple labeling* and denote the set of candidate simple labelings $C^{sv} = \{c_k^{sv}\}_k$. Note that a given pair t_i^{sv}, g_j^v may generate more than one simple labeling, if there is more than one way to assign the elements of g_j^v to constituents in t_i^{sv}. Also, for a sentence s there may be several simple labelings that lead to the same role labeling. In particular, there may be several simple labelings which assign the correct labels to all constituents; we denote this set $K^{sv} \subseteq C^{sv}$.

6 Probabilistic Model

We now define our probabilistic model. Given a (possibly large) set of candidate simple labelings C^{sv}, we need to select a correct one. We assign a score to each candidate based on its features:

Rule = Depassivize
Pattern = {ARG0 = Subj NP, ARG1 = PV NP2, ARG2 = PV NP1}
Role = ARG0, Head Word = John
Role = ARG1, Head Word = sandwich
Role = ARG2, Head Word = I
Role = ARG0, Category = NP
Role = ARG1, Category = NP
Role = ARG2, Category = NP
Role = ARG0, Position = Subject NP
Role = ARG1, Position = Postverb NP2
Role = ARG2, Position = Postverb NP1

Figure 6: Features for "John gave me a sandwich."

which rules were used to obtain the simple sentence, which role pattern was used, and features about the assignment of constituents to roles. A log-linear model then assigns probability to each simple labeling equal to the normalized exponential of the score.

The first type of feature is which rules were used to obtain the simple sentence. These features are indicator functions for each possible rule. Thus, we do not currently learn anything about interactions between different rules. The second type of feature is an indicator function of the role pattern used to generate the labeling. This allows us to learn that "give" has a preference for the labeling $\{$*ARG0 = Subject NP, ARG1 = Postverb NP2, ARG2 = Postverb NP1*$\}$. Our final features are analogous to those used in semantic role labeling, but greatly simplified due to our use of simple sentences: head word of the constituent; category (i.e., constituent label); and position in the simple sentence. Each of these features is combined with the role assignment, so that each feature indicates a preference for a particular role assignment (i.e., for "give", head word "sandwich" tends to be ARG1). For each feature, we have a verb-specific and a verb-independent version, allowing sharing across verbs while still permitting different verbs to learn different preferences. The set of extracted features for the sentence "I was given a sandwich by John" with simplification "John gave me a sandwich" is shown in Figure 6. We omit verb-specific features to save space . Note that we "stem" all pronouns (including possessive pronouns).

For each candidate simple labeling c_k^{sv} we extract a vector of features $\mathbf{f}_k^{sv}$ as described above. We now define the probability of a simple labeling c_k^{sv} with respect to a weight vector $\mathbf{w}$ $P(c_k^{sv}) = \frac{e^{\mathbf{w}^T \mathbf{f}_k^{sv}}}{\sum_{k'} e^{\mathbf{w}^T \mathbf{f}_{k'}^{sv}}}$.

Our goal is to maximize the total probability assigned to any correct simple labeling; therefore, for each sentence/verb pair (s, v), we want to increase $\sum_{c_k^{sv} \in K^{sv}} P(c_k^{sv})$. This expression treats the simple labeling (consisting of a simple sentence and a role assignment) as a hidden variable that is summed out. Taking the log, summing across all sentence/verb pairs, and adding L2 regularization on the weights, we have our final objective $F(\mathbf{w})$:

$$\sum_{s,v} \left(\log \frac{\sum_{c_k^{sv} \in K^{sv}} e^{\mathbf{w}^T \mathbf{f}_k^{sv}}}{\sum_{c_{k'}^{sv} \in C^{sv}} e^{\mathbf{w}^T \mathbf{f}_{k'}^{sv}}} \right) - \frac{\mathbf{w}^T \mathbf{w}}{2\sigma^2}$$

We train our model by optimizing the objective using standard methods, specifically BFGS. Due to the summation over the hidden variable representing the choice of simple sentence (not observed in the training data), our objective is not convex. Thus, we are not guaranteed to find a global optimum; in practice we have gotten good results using the default initialization of setting all weights to 0.

Consider the derivative of the likelihood component with respect to a single weight w_l:

$$\sum_{c_k^{sv} \in K^{sv}} \mathbf{f}_k^{sv}(l) \frac{P(c_k^{sv})}{\sum_{c_{k'}^{sv} \in K^{sv}} P(c_{k'}^{sv})} - \sum_{c_k^{sv} \in C^{sv}} \mathbf{f}_k^{sv}(l) P(c_k^{sv})$$

where $\mathbf{f}_k^{sv}(l)$ denotes the l^{th} component of $\mathbf{f}_k^{sv}$. This formula is positive when the expected value of the l^{th} feature is higher on the set of correct simple labelings K^{sv} than on the set of all simple labelings C^{sv}. Thus, the optimization procedure will tend to be self-reinforcing, increasing the score of correct simple labelings which already have a high score.

7 Simplification Data Structure

Our representation of the set of possible simplifications of a sentence addresses two computational bottlenecks. The first is the need to repeatedly copy large chunks of the sentence. For example, if we are depassivizing a sentence, we can avoid copying the subject and object of the original sentence by simply referring back to them in the depassivized version. At worst, we only need to add one node for each numbered node in the transformation rule. The second issue is the possible exponential blowup of the number of generated sentences. Consider "I want to eat and I want to drink and I want to play and ..." Each subsentence can be simplified, yielding two possibilities for each subsentence. The number of simplifications of the entire sentence is then exponential in the length of the sentence. However,

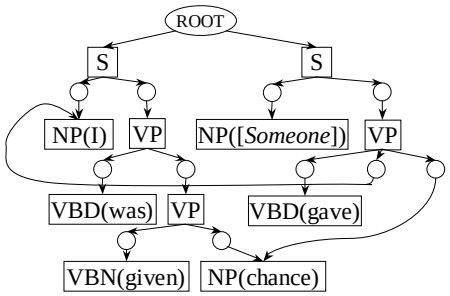

Figure 7: Data structure after applying the depassivize rule to "I was given (a) chance." Circular nodes are OR-nodes, rectangular nodes are AND-nodes.

we can store these simplifications compactly as a set of independent decisions, "I {want to eat OR eat} and I {want to drink OR drink} and ..."

Both issues can be addressed by representing the set of simplifications using an AND-OR tree, a general data structure also used to store parse forests such as those produced by a chart parser. In our case, the AND nodes are similar to constituent nodes in a parse tree – each has a category (e.g. NP) and (if it is a leaf) a word (e.g. "chance"), but instead of having a list of child constituents, it instead has a list of child OR nodes. Each OR node has one or more constituent children that correspond to the different options at this point in the tree. Figure 7 shows the resulting AND-OR tree after applying the depassivize rule to the original parse of "I was given a chance." Because this AND-OR tree represents only two different parses, the original parse and the depassivized version, only one OR node in the tree has more than one child – the root node, which has two choices, one for each parse. However, the AND nodes immediately above "I" and "chance" each have more than one OR-node parent, since they are shared by the original and depassivized parses[1]. To extract a parse from this data structure, we apply the following recursive algorithm: starting at the root OR node, each time we reach an OR node, we choose and recurse on exactly one of its children; each time we reach an AND node, we recurse on all of its children. In Figure 7, we have only one choice: if we go left at the root, we generate the original parse; otherwise, we generate the depassivized version.

Unfortunately, it is difficult to find the optimal AND-OR tree. We use a greedy but smart procedure to try to produce a small tree. We omit details for lack of space. Using our rule set, the compact representation is usually (but not always) small.

For our compact representation to be useful, we need to be able to optimize our objective without expanding all possible simple sentences. A relatively straight-forward extension of the inside-outside algorithm for chart-parses allows us to learn and perform inference in our compact representation (a similar algorithm is presented in (Geman & Johnson, 2002)). We omit details for lack of space.

8 Experiments

We evaluated our system using the setup of the Conll 2005 semantic role labeling task.[2] Thus, we trained on Sections 2-21 of PropBank and used Section 24 as development data. Our test data includes both the selected portion of Section 23 of PropBank, plus the extra data on the Brown corpus. We used the Charniak parses provided by the Conll distribution.

We compared to a strong **Baseline** SRL system that learns a logistic regression model using the features of Pradhan et al. (2005). It has two stages. The first filters out nodes that are unlikely to be arguments. The second stage labels each remaining node either as a particular role (e.g. "ARGO") or as a non-argument. Note that the baseline feature set includes a feature corresponding to the subcategorization of the verb (specifically, the sequence of nonterminals which are children of the predicate's parent node). Thus, **Baseline** does have access to something similar to our model's role pattern feature, although the **Baseline** subcategorization feature only includes post-verbal modifiers and is generally much noisier because it operates on the original sentence.

Our **Transforms** model takes as input the Charniak parses supplied by the Conll release, and labels every node with Core arguments (ARG0-ARG5). Our rule set does not currently handle either referent arguments (such as "who" in "The man who ate ...") or non-core arguments (such as ARGM-TMP). For these arguments, we simply filled in using our baseline system (specifically, any non-core argument which did not overlap an argument predicted by our model was added to the labeling). Also, on some sentences, our system did not generate any predictions because no valid simple sen-

[1] In this particular example, both of these nodes are leaves, but in general shared nodes can be entire tree fragments

[2] http://www.lsi.upc.es/ srlconll/home.html

Model	Dev	Test WSJ	Test Brown	Test WSJ+Br
Baseline	74.7	76.9	64.7	75.3
Transforms	75.6	77.4	**66.8**	76.0
Combined	**76.0**	**78.0**	66.4	**76.5**
Punyakanok	77.35	79.44	67.75	77.92

Table 2: F1 Measure using Charniak parses

tences were produced by the simplification system . Again, we used the baseline to fill in predictions (for all arguments) for these sentences.

Baseline and **Transforms** were regularized using a Gaussian prior; for both models, $\sigma^2 = 1.0$ gave the best results on the development set.

For generating role predictions from our model, we have two reasonable options: use the labeling given by the single highest scoring simple labeling; or compute the distribution over predictions for each node by summing over all simple labelings. The latter method worked slightly better, particularly when combined with the baseline model as described below, so all reported results use this method.

We also evaluated a hybrid model that combines the **Baseline** with our simplification model. For a given sentence/verb pair (s, v), we find the set of constituents N^{sv} that made it past the first (filtering) stage of **Baseline**. For each candidate simple sentence/labeling pair $c_k^{sv} = (t_i^{sv}, g_j^v)$ proposed by our model, we check to see which of the constituents in N^{sv} are already present in our simple sentence t_i^{sv}. Any constituents that are *not* present are then assigned a probability distribution over possible roles according to **Baseline**. Thus, we fall back **Baseline** whenever the current simple sentence does not have an "opinion" about the role of a particular constituent. The **Combined** model is thus able to correctly label sentences when the simplification process drops some of the arguments (generally due to unusual syntax). Each of the two components was trained separately and combined only at testing time.

Table 2 shows results of these three systems on the Conll-2005 task, plus the top-performing system (Punyakanok et al., 2005) for reference. **Baseline** already achieves good performance on this task, placing at about 75^{th} percentile among evaluated systems. Our **Transforms** model outperforms **Baseline** on all sets. Finally, our **Combined** model improves over **Transforms** on all but the test Brown corpus,

Model	Test WSJ
Baseline	87.6
Transforms	88.2
Combined	**88.5**

Table 3: F1 Measure using gold parses

achieving a statistically significant increase over the **Baseline** system (according to confidence intervals calculated for the Conll-2005 results).

The **Combined** model still does not achieve the performance levels of the top several systems. However, these systems all use information from multiple parses, allowing them to fix many errors caused by incorrect parses. We return to this issue in Section 10. Indeed, as shown in Table 3, performance on gold standard parses is (as expected) much better than on automatically generated parses, for all systems. Importantly, the **Combined** model again achieves a significant improvement over **Baseline**.

We expect that by labeling simple sentences, our model will generalize well even on verbs with a small number of training examples. Figure 8 shows F1 measure on the WSJ test set as a function of training set size. Indeed, both the **Transforms** model and the **Combined** model significantly outperform the **Baseline** model when there are fewer than 20 training examples for the verb. While the **Baseline** model has higher accuracy than the **Transforms** model for verbs with a very large number of training examples, the **Combined** model is at or above both of the other models in all but the rightmost bucket, suggesting that it gets the best of both worlds.

We also found, as expected, that our model improved on sentences with very long parse paths. For example, in the sentence "Big investment banks refused to step up to the plate to support the beleagured floor traders by buying blocks of stock, traders say," the parse path from "buy" to its ARG0, "Big investment banks," is quite long. The **Transforms** model correctly labels the arguments of "buy", while the **Baseline** system misses the ARG0.

To understand the importance of different types of rules, we performed an ablation analysis. For each major rule category in Figure 1, we deleted those rules from the rule set, retrained, and evaluated using the **Combined** model. To avoid parse-related issues, we trained and evaluated on gold-standard parses. Most important were rules relating to (ba-

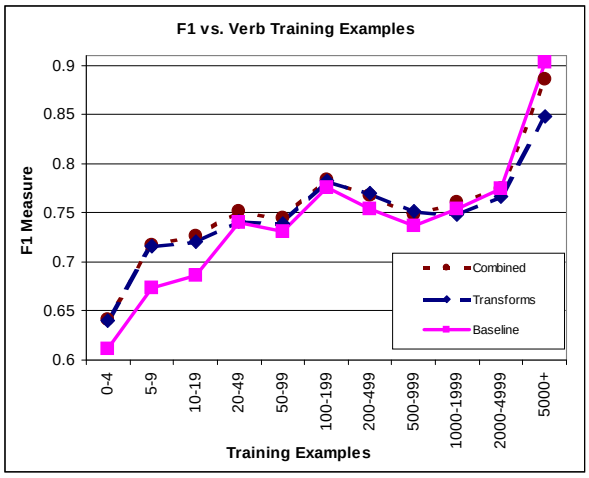

Figure 8: F1 Measure on the WSJ test set as a function of training set size. Each bucket on the X-axis corresponds to a group of verbs for which the number of training examples fell into the appropriate range; the value is the average performance for verbs in that bucket.

sic) verb raising/control, "make" rewrites, modified nouns, and passive constructions. Each of these rule categories when removed lowered the F1 score by approximately .4%. In constrast, removing rules for non-basic control, possessives, and inverted sentences caused a negligible reduction in performance. This may be because the relevant syntactic structures occur rarely; because **Baseline** does well on those constructs; or because the simplification model has trouble learning when to apply these rules.

9 Related Work

One area of current research which has similarities with this work is on Lexical Functional Grammars (LFGs). Both approaches attempt to abstract away from the surface level syntax of the sentence (e.g., the XLE system[3]). The most obvious difference between the approaches is that we use SRL data to train our system, avoiding the need to have labeled data specific to our simplification scheme.

There have been a number of works which model verb subcategorization. Approaches include incorporating a subcategorization feature (Gildea & Jurafsky, 2002; Xue & Palmer, 2004), such as the one used in our baseline; and building a model which jointly classifies all arguments of a verb (Toutanova et al., 2005). Our method differs from past work in that it extracts its role pattern feature from the simplified sentence. As a result, the feature is less noisy and generalizes better across syntactic variation than a feature extracted from the original sentence.

Another group of related work focuses on summarizing sentences through a series of deletions (Jing, 2000; Dorr et al., 2003; Galley & McKeown, 2007). In particular, the latter two works iteratively simplify the sentence by deleting a phrase at a time. We differ from these works in several important ways. First, our transformation language is not context-free; it can reorder constituents and then apply transformation rules to the reordered sentence. Second, we are focusing on a somewhat different task; these works are interested in obtaining a single summary of each sentence which maintains all "essential" information, while in our work we produce a simplification that may lose semantic content, but aims to contain all arguments of a verb. Finally, training our model on SRL data allows us to avoid the relative scarcity of parallel simplification corpora and the issue of determining what is "essential" in a sentence.

Another area of related work in the semantic role labeling literature is that on tree kernels (Moschitti, 2004; Zhang et al., 2007). Like our method, tree kernels decompose the parse path into smaller pieces for classification. Our model can generalize better across verbs because it first simplifies, then classifies based on the simplified sentence. Also, through iterative simplifications we can discover structure that is not immediately apparent in the original parse.

10 Future Work

There are a number of improvements that could be made to the current simplification system, including augmenting the rule set to handle more constructions and doing further sentence normalizations, e.g., identifying whether a direct object exists. Another interesting extension involves incorporating parser uncertainty into the model; in particular, our simplification system is capable of seamlessly accepting a parse forest as input.

There are a variety of other tasks for which sentence simplification might be useful, including summarization, information retrieval, information extraction, machine translation and semantic entailment. In each area, we could either use the simplification system as learned on SRL data, or retrain the simplification model to maximize performance on the particular task.

[3] http://www2.parc.com/isl/groups/nltt/xle/

References

Dorr, B., Zajic, D., & Schwartz, R. (2003). Hedge: A parse-and-trim approach to headline generation. *Proceedings of the HLT-NAACL Text Summarization Workshop and Document Understanding Conference*.

Galley, M., & McKeown, K. (2007). Lexicalized markov grammars for sentence compression. *Proceedings of NAACL-HLT*.

Geman, S., & Johnson, M. (2002). Dynamic programming for parsing and estimation of stochastic unification-based grammars. *Proceedings of ACL*.

Gildea, D., & Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics*.

Jing, H. (2000). Sentence reduction for automatic text summarization. *Proceedings of Applied NLP*.

Moschitti, A. (2004). A study on convolution kernels for shallow semantic parsing. *Proceedings of ACL*.

Pradhan, S., Hacioglu, K., Krugler, V., Ward, W., Martin, J. H., & Jurafsky, D. (2005). Support vector learning for semantic argument classification. *Machine Learning*, *60*, 11–39.

Punyakanok, V., Koomen, P., Roth, D., & Yih, W. (2005). Generalized inference with multiple semantic role labeling systems. *Proceedings of CoNLL*.

Toutanova, K., Haghighi, A., & Manning, C. (2005). Joint learning improves semantic role labeling. *Proceedings of ACL*, 589–596.

Xue, N., & Palmer, M. (2004). Calibrating features for semantic role labeling. *Proceedings of EMNLP*.

Zhang, M., Che, W., Aw, A., Tan, C. L., Zhou, G., Liu, T., & Li, S. (2007). A grammar-driven convolution tree kernel for semantic role classification. *Proceedings of ACL*.