Solving the Problem of Cascading Errors: Approximate Bayesian Inference for Linguistic Annotation Pipelines

Jenny Rose Finkel, Christopher D. Manning and Andrew Y. Ng
Computer Science Department
Stanford University
Stanford, CA 94305
{jrfinkel, manning, ang}@cs.stanford.edu

Abstract

The end-to-end performance of natural language processing systems for compound tasks, such as question answering and textual entailment, is often hampered by use of a greedy 1-best pipeline architecture, which causes errors to propagate and compound at each stage. We present a novel architecture, which models these pipelines as Bayesian networks, with each low level task corresponding to a variable in the network, and then we perform approximate inference to find the best labeling. Our approach is extremely simple to apply but gains the benefits of sampling the entire distribution over labels at each stage in the pipeline. We apply our method to two tasks – semantic role labeling and recognizing textual entailment – and achieve useful performance gains from the superior pipeline architecture.

1 Introduction

Almost any system for natural language understanding must recover hidden linguistic structure at many different levels: parts of speech, syntactic dependencies, named entities, etc. For example, modern semantic role labeling (SRL) systems use the parse of the sentence, and question answering requires question type classification, parsing, named entity tagging, semantic role labeling, and often other tasks, many of which are dependent on one another and must be pipelined together. Pipelined systems are ubiquitous in NLP: in addition to the above examples, commonly parsers and named entity recognizers use part of speech tags and chunking information, and also word segmentation for languages such as Chinese. Almost no NLP task is truly standalone.

Most current systems for higher-level, aggregate NLP tasks employ a simple 1-best feed forward architecture: they greedily take the best output at each stage in the pipeline and pass it on to the next stage. This is the simplest architecture to build (particularly if reusing existing component systems), but errors are frequently made during this pipeline of annotations, and when a system is given incorrectly labeled input it is much harder for that system to do its task correctly. For example, when doing semantic role labeling, if no syntactic constituent of the parse actually corresponds to a given semantic role, then that semantic role will almost certainly be misidentified. It is therefore disappointing, but not surprising, that F-measures on SRL drop more than 10% when switching from gold parses to automatic parses (for instance, from 91.2 to 80.0 for the joint model of Toutanova (2005)).

A common improvement on this architecture is to pass k-best lists between processing stages, for example (Sutton and McCallum, 2005; Wellner et al., 2004). Passing on a k-best list gives useful improvements (e.g., in Koomen et al. (2005)), but efficiently enumerating k-best lists often requires very substantial cognitive and engineering effort, e.g., in (Huang and Chiang, 2005; Toutanova et al., 2005).

At the other extreme, one can maintain the entire space of representations (and their probabilities) at each level, and use this full distribution to calculate the full distribution at the next level. If restricting oneself to weighted finite state transducers (WFSTs), a framework applicable to a number of NLP applications (as outlined in Karttunen (2000)), a pipeline can be compressed down into a single WFST, giving outputs equivalent to propagating the entire distribution through the pipeline. In the worst case there is an exponential space cost, but in many relevant cases composition is in practice quite practical. Outside of WFSTs, maintaining entire probability distributions is usually infeasible in NLP, because for most intermediate tasks, such as parsing and named entity recognition, there is an exponential number of possible labelings. Nevertheless, for some models, such as most parsing models, these exponential labelings can be compactly represented in a packed form, e.g., (Maxwell and Kaplan, 1995; Crouch, 2005), and subsequent stages can be reengineered to work over these packed representations, e.g., (Geman and Johnson, 2002). However, doing this normally also involves a very high cognitive and engineering effort, and in practice this solution is infrequently adopted. Moreover, in some cases, a subsequent module is incompatible with the packed representation of a previous module and an exponential amount of work is nevertheless required within this architecture.

Here we present an attractive middle ground in dealing with linguistic pipelines. Rather than only using the 1 or k most likely labelings at each stage, we would indeed like to take into account all possible labelings and their probabilities, but we would like to be able to do so without a lot of thinking or engineering. We propose that this can be achieved by use of approximate inference. The form of approximate inference we use is very simple: at each stage in the pipeline, we draw a sample from the distribution of labels, conditioned on the samples drawn at previous stages. We repeat this many times, and then use the samples from the last stage, which corresponds to the ultimate, higher-level task, to form a majority vote classifier. As the number of samples increases, this method will approximate the complete distribution. Use of the method is normally a simple modification to an existing piece of code, and the method is general. It can be applied not only to all pipelines, but to multi-stage algorithms which are not pipelines as well.

We apply our method to two problems: semantic role labeling and recognizing textual entailment. For semantic role labeling we use a two stage pipeline which parses the input sentence, and for recognizing textual entailment we use a three stage pipeline which tags the sentence with named entities and then parses it before passing it to the entailment decider.

2 Approach

2.1 Overview

In order to do approximate inference, we model the entire pipeline as a Bayesian network. Each stage in the pipeline corresponds to a variable in the network. For example, the parser stage corresponds to a variable whose possible values are all possible parses of the sentence. The probabilities of the parses are conditioned on the parent variables, which may just be the words of the sentence, or may be the part of speech tags output by a part of speech tagger.

The simple linear structure of a typical linguistic annotation network permits exact inference that is quadratic in the number of possible labels at each stage, but unfortunately our annotation variables have a very large domain. Additionally, some networks may not even be linear; frequently one stage may require the output from multiple previous stages, or multiple earlier stages may be completely independent of one another. For example, a typical QA system will do question type classification on the question, and from that extract keywords which are passed to the information retreival part of the system. Meanwhile, the retreived documents are parsed and tagged with named entities; the network rejoins those outputs with the question type classification to decide on the correct answer. We address these issues by using approximate inference instead of exact inference. The structure of the nodes in the network permits direct sampling based on a topological sort of the nodes. Samples are drawn from the conditional distributions of each node, conditioned on the samples drawn at earlier nodes in the topological sort.

2.2 Probability of a Complete Labeling

Before we can discuss how to sample from these Bayes nets, we will formalize how to move from an annotation pipeline to a Bayes net. Let $\mathbf{A}$ be the set of n annotators A_1, A_2, ..., A_n (e.g., part of speech tagger, named entity recognizer, parser). These are the variables in the network. For annotator a_i, we denote the set of other annotators whose input is directly needed as $Parents(A_i) \subset \mathbf{A}$ and a particular assignment to those variables is $parents(A_i)$. The possible values for a particular annotator A_i are a_i (e.g., a particular parse tree or named entity tagging). We can now formulate the probability of a complete annotation (over all annotators) in the standard way for Bayes nets:

$$\mathrm{P}_{\mathrm{BN}}(a_1, a_2, ..., a_n) = \prod_{i=1}^{N} P(a_i | parents(A_i)) \tag{1}$$

2.3 Approximate Inference in Bayesian Networks

This factorization of the joint probability distribution facilitates inference. However, exact inference is intractable because of the number of possible values for our variables. Parsing, part of speech tagging, and named entity tagging (to name a few) all have a number of possible labels that is exponential in the length of the sentence, so we use approximate inference. We chose Monte Carlo inference, in which samples drawn from the joint distribution are used to approximate a marginal distribution for a subset of variables in the distribution. First, the nodes are sorted in topological order. Then, samples are drawn for each variable, conditioned on the samples which have already been drawn. Many samples are drawn, and are used to estimate the joint distribution.

Importantly, for many language processing tasks our application only needs to provide the most likely value for a high-level linguistic annotation (e.g., the guessed semantic roles, or answer to a question), and other annotations such as parse trees are only present to assist in performing that task. The probability of the final annotation is given by:

$$\mathrm{P}_{\mathrm{BN}}(a_n) = \sum_{a_1, a_2, ..., a_{n-1}} \mathrm{P}_{\mathrm{BN}}(a_1, a_2, ..., a_n) \tag{2}$$

Because we are summing out all variables other than the final one, we effectively use only the samples drawn from the final stage, ignoring the labels of the variables, to estimate the marginal distribution over that variable. We then return the label which had the highest number of samples. For example, when trying to recognize textual entailment, we count how many times we sampled "yes, it is entailed" and how many times we sampled "no, it is not entailed" and return the answer with more samples.

When the outcome you are trying to predict is binary (as is the case with RTE) or n-ary for small n, the number of samples needed to obtain a good estimate of the posterior probability is very small. This is true even if the spaces being sampled from during intermediate stages are exponentially large (such as the space of all parse trees). Ng and Jordan (2001) show that under mild assumptions, with only N samples the relative classification error will be at most $O(\frac{1}{N})$ higher than the error of the Bayes optimal classifier (in our case, the classifier which does exact inference). Even if the outcome space is not small, the sampling technique we present can still be very useful, as we will see later for the case of SRL.

3 Generating Samples

The method we have outlined requires the ability to sample from the conditional distributions in the factored distribution of (1): in our case, the probability of a particular linguistic annotation, conditioned on other linguistic annotations. Note that this differs from the usual annotation task: taking the argmax. But for most algorithms the change is a small and easy change. We discuss how to obtain samples efficiently from a few different annotation models: probabilistic context free grammars (PCFGs), and conditional random fields (CRFs).

3.1 Sampling Parses

Bod (1995) discusses parsing with probabilistic tree substitution grammars, which, unlike simple PCFGs, do not have a one-to-one mapping between output parse trees and a derivation (a bag of rules) that produced it, and hence the most-likely derivation may not correspond to the most likely parse tree. He therefore presents a bottom-up approach to sampling derivations from a derivation forest, which does correspond to a sample from the space of parse trees. Goodman (1998) presents a top-down version of this algorithm. Although we use a PCFG for parsing, it is the grammar of (Klein and Manning, 2003), which uses extensive state-splitting, and so there is again a many-to-one correspondence between derivations and parses, and we use an algorithm similar to Goodman's in our work.

PCFGs put probabilities on each rule, such as *S* → *NP VP* and *NN* → *'dog'*. The probability of a parse is the product of the probabilities of the rules used to construct the parse tree. A dynamic programing algorithm, the *inside algorithm*, can be used to find the probability of a sentence. The *inside probability* $\beta_k(p,q)$ is the probability that words p through q, inclusive, were produced by the non-terminal k. So the probability of the sentence *The boy pet the dog.* is equal to the inside probability $\beta_S(1,6)$, where the first word, w_1 is *The* and the sixth word, w_6, is *[period]*. It is also useful for our purposes to view this quantity as the sum of the probabilities of all parses of the sentence which have *S* as the start symbol. The probability can be defined recursively (Manning and Schütze, 1999) as follows:

$$\beta_k(p,q) = \begin{cases} P(N^k \rightarrow w_p) \quad \text{if} \quad p = q \\ \sum_{r,s} \sum_{d=p}^{q-1} P(N^k \rightarrow N^r N^s)\beta_r(p,d)\beta_s(d+1,q) \\ \qquad\qquad \text{otherwise} \end{cases} \tag{3}$$

where N^k, N^r and N^s are non-terminal symbols and w_p is the word at position p. We have omitted the case of unary rules for simplicity since it requires a closure operation.

These probabilities can be efficiently computed using a dynamic program. or memoization of each value as it is calculated. Once we have computed all of the inside probabilities, they can be used to generate parses from the distribution of all parses of the sentence, using the algorithm in Figure 1.

This algorithm is called after all of the inside probabilities have been calculated and stored, and take as parameters S, 1, and $length(sentence)$. It works by building the tree, starting from the root, and recursively generating children based on the posterior probabilities of applying each rule and each possible position on which to split the sentences. Intuitively, the algorithm is given a non-terminal symbol, such as *S* or *NP*, and a span of words, and has to decide (a) what rule to apply to expand the non-terminal, and (b) where to split the span of words, so that each non-terminal resulting from applying the rule has an associated word span, and the process can repeat. The inside probabilities are calculated just once, and we can then generate many samples very quickly; *DrawSamples* is linear in the number of words, and rules.

3.2 Sampling Named Entity Taggings

To do named entity recognition, we chose to use a conditional random field (CRF) model, based on Lafferty et al. (2001). CRFs represent the state of the art in sequence modeling – they are discriminatively trained, and maximize the joint likelihood of the entire label sequence in a manner which allows for bi-directional flow of information. In order to describe how samples are generated, we generalize CRFs in a way that is consistent with the Markov random field literature. We create a linear chain of *cliques*, each of which represents the probabilistic relationship between an adjacent set of n states using a *factor table* containing $|S|^n$ values. These factor tables on their own should *not* be viewed as probabilities, unnormalized or otherwise. They are, however, defined in terms of exponential models conditioned on features of the observation sequence, and must be instantiated for each new observation sequence. The probability of a state sequence is then defined by the sequence of factor tables in the clique chain, given the observation sequence:

```
function DRAWSAMPLE($N^k, r, s$)
 if $r = s$
   $tree.label = N^k$
   $tree.child = word(r)$
   return $(tree)$
 for each rule $m \in \{m' : head(m') = N^k\}$
   $N^i \leftarrow lChild(m)$
   $N^j \leftarrow rChild(m)$
   for $q \leftarrow r$ to $s - 1$
     $scores(m,q) \leftarrow P(m)\beta_i(r,q)\beta_j(q+1,s)$
 $(m,q) \leftarrow$ SAMPLEFROM($scores$)
 $tree.label = head(m)$
 $tree.lChild =$ DRAWSAMPLE($lChild(m), r, q$)
 $tree.rChild =$ DRAWSAMPLE($rChild(m), q+1, s$)
 return $(tree)$
```

Figure 1: Pseudo-code for sampling parse trees from a PCFG. This is a recursive algorithm which starts at the root of the tree and expands each node by sampling from the distribution of possible rules and ways to split the span of words. Its arguments are a non-terminal and two integers corresponding to word indices, and it is initially called with arguments *S*, 1, and the length of the sentence. There is a call to *sampleFrom*, which takes an (unnormalized) probability distribution, normalizes it, draws a sample and then returns the sample.

$$\mathrm{P}_{\mathrm{CRF}}(\mathbf{s}|\mathbf{o}) = \frac{1}{Z(\mathbf{o})} \prod_{i=1}^{N} F_i(s_{i-n} \ldots s_i) \tag{4}$$

where $F_i(s_{i-n} \ldots s_i)$ is the element of the factor table at position i corresponding to states s_{i-n} through s_i, and $Z(o)$ is the partition function which serves to normalize the distribution.[1] To in-

[1] To handle the start condition properly, imagine also that we define a set of distinguished start states $s_{-(n-1)} \ldots s_0$.

fer the most likely state sequence in a CRF it is customary to use the Viterbi algorithm.

We then apply a process called *clique tree calibration*, which involves passing *messages* between the cliques (see Cowell et al. (2003) for a full treatment of this topic). After this process has completed, the factor tables can be viewed as unnormalized probabilities, which can be used to compute conditional probabilities, $\mathrm{P}_{\mathrm{CRF}}(s_i|s_{i-n}\ldots s_{i-1},o)$. Once these probabilities have been calculated, generating samples is very simple. First, we draw a sample for the label at the first position,[2] and then, for each subsequent position, we draw a sample from the distribution for that position, conditioned on the label sampled at the previous position. This process results in a sample of a complete labeling of the sequence, drawn from the posterior distribution of complete named entity taggings.

Similarly to generating sample parses, the expensive part is calculating the probabilities; once we have them we can generate new samples very quickly.

3.3 k-Best Lists

At first glance, k-best lists may seem like they should outperform sampling, because in effect they are the k best samples. However, there are several important reasons why one might prefer sampling. One reason is that the k best paths through a word lattice, or the k best derivations in parse forest do not necessarily correspond to the k best sentences or parse trees. In fact, there are no known sub-exponential algorithms for the best outputs in these models, when there are multiple ways to derive the same output.[3] This is not just a theoretical concern – the Stanford parser uses such a grammar, and we found that when generating a 50-best derivation list that on average these derivations corresponded to about half as many unique parse trees. Our approach circumvents this issue entirely, because the samples are generated from the actual output distribution.

Intuition also suggests that sampling should give more diversity at each stage, reducing the likelihood of not even considering the correct output. Using the Brown portion of the SRL test set (discussed in sections 4 and 6.1), and 50-samples/50-best, we found that on average the 50-samples system considered approximately 25% more potential SRL labelings than the 50-best system.

When pipelines have more than two stages, it is customary to do a beam search, with a beam size of k. This means that at each stage in the pipeline, more and more of the probability mass gets "thrown away." Practically, this means that as pipeline length increases, there will be increasingly less diversity of labels from the earlier stages. In a degenerate 10-stage, k-best pipeline, where the last stage depends mainly on the first stage, it is probable that all but a few labelings from the first stage will have been pruned away, leaving something much smaller than a k-best sample, possibly even a 1-best sample, as input to the final stage. Using approximate inference to estimate the marginal distribution over the last stage in the pipeline, such as our sampling approach, the pipeline length does not have this negative impact or affect the number of samples needed. And unlike k-best beam searches, there is an entire research community, along with a large body of literature, which studies how to do approximate inference in Bayesian networks and can provide performance bounds based on the method and the number of samples generated.

One final issue with the k-best method arises when instead of a linear chain pipeline, one is using a general directed acyclic graph where a node can have multiple parents. In this situation, doing the k-best calculation actually becomes exponential in the size of the largest in-degree of a node – for a node with n parents, you must try all k^n combinations of the values for the parent nodes. With sampling this is not an issue; each sample can be generated based on a topological sort of the graph.

4 Semantic Role Labeling

4.1 Task Description

Given a sentence and a target verb (the *predicate*) the goal of semantic role labeling is to identify and label syntactic constituents of the parse tree with semantic roles of the predicate. Common roles are *agent*, which is the thing performing the action, *patient*, which is the thing on which the action is being performed, and *instrument*, which is the thing with which the action is being done. Additionally, there are *modifier arguments* which can specify the location, time, manner, etc. The following sentence provides an example of a predi-

[2] Conditioned on the distinguished start states.

[3] Many thanks to an anonymous reviewer for pointing out this argument.

cate and its arguments:

> [The luxury auto maker]$_{agent}$ [last year]$_{temp}$ [sold]$_{pred}$ [1,214 cars]$_{patient}$ in [the U.S]$_{location}$.

Semantic role labeling is a key component for systems that do question answering, summarization, and any other task which directly uses a semantic interpretation.

4.2 System Description

We modified the system described in Haghighi et al. (2005) and Toutanova et al. (2005) to test our method. The system uses both local models, which score subtrees of the entire parse tree independently of the labels of other nodes not in that subtree, and joint models, which score the entire labeling of a tree with semantic roles (for a particular predicate).

First, the task is separated into two stages, and local models are learned for each. At the first stage, the *identification stage*, a classifier labels each node in the tree as either *ARG*, meaning that it is an argument (either core or modifier) to the predicate, or *NONE*, meaning that it is not an argument. At the second stage, the *classification stage*, the classifier is given a set of arguments for a predicate and must label each with its semantic role.

Next, a Viterbi-like dynamic algorithm is used to generate a list of the k-best joint (identification and classification) labelings according to the local models. The algorithm enforces the constraint that the roles should be non-overlapping. Finally, a joint model is constructed which scores a completely labeled tree, and it is used to re-rank the k-best list. The separation into local and joint models is necessary because there are an exponential number of ways to label the entire tree, so using the joint model alone would be intractable. Ideally, we would want to use approximate inference instead of a k-best list here as well. Particle filtering would be particularly well suited - particles could be sampled from the local model and then reweighted using the joint model. Unfortunately, we did not have enough time modify the code of (Haghighi et al., 2005) accordingly, so the k-best structure remained.

To generate samples from the SRL system, we take the scores given to the k-best list, normalize them to sum to 1, and sample from them. One consequence of this, is that any labeling not on the k-best list has a probability of 0.

5 Recognizing Textual Entailment

5.1 Task Description

In the task of recognizing textual entailment (RTE), also commonly referred to as robust textual inference, you are provided with two passages, a *text* and a *hypothesis*, and must decide whether the hypothesis can be inferred from the text. The term *robust* is used because the task is not meant to be domain specific. The term *inference* is used because this is not meant to be logical entailment, but rather what an intelligent, informed human would infer. Many NLP applications would benefit from the ability to do robust textual entailment, including question answering, information retrieval and multi-document summarization. There have been two PASCAL workshops (Dagan et al., 2005) with shared tasks in the past two years devoted to RTE. We used the data from the 2006 workshop, which contains 800 text-hypothesis pairs in each of the test and development sets[4] (there is no training set). Here is an example from the development set from the first RTE challenge:

Text: *Researchers at the Harvard School of Public Health say that people who drink coffee may be doing a lot more than keeping themselves awake – this kind of consumption apparently also can help reduce the risk of diseases.*

Hypothesis: *Coffee drinking has health benefits.*

The positive and negative examples are balanced, so the baseline of guessing either all *yes* or all *no* would score 50%. This is a hard task – at the first challenge no system scored over 60%.

5.2 System Description

MacCartney et al. (2006) describe a system for doing robust textual inference. They divide the task into three stages – linguistic analysis, graph alignment, and entailment determination. The first of these stages, *linguistic analysis* is itself a pipeline of parsing and named entity recognition. They use the syntactic parse to (deterministically) produce a typed dependency graph for each sentence. This pipeline is the one we replace. The second stage, *graph alignment* consists of trying to find good alignments between the typed dependency graphs

[4] The dataset and further information from both challenges can be downloaded from *http://www.pascal-network.org/Challenges/RTE2/Datasets/*

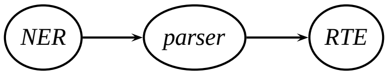

Figure 2: The pipeline for recognizing textual entailment.

for the text and hypothesis. Each possible alignment has a score, and the alignment with the best score is propagated forward. The final stage, *entailment determination*, is where the decision is actually made. Using the score from the alignment, as well as other features, a logistic model is created to predict entailment. The parameters for this model are learned from development data.[5] While it would be preferable to sample possible alignments, their system for generating alignment scores is not probabilistic, and it is unclear how one could convert between alignment scores and probabilities in a meaningful way.

Our modified linguistic analysis pipeline does NER tagging and parsing (in their system, the parse is dependent on the NER tagging because some types of entities are pre-chunked before parsing) and treats the remaining two sections of their pipeline, the alignment and determination stages, as one final stage. Because the entailment determination stage is based on a logistic model, a probability of entailment is given and sampling is straightforward.

6 Experimental Results

In our experiments we compare the greedy pipelined approach with our sampling pipeline approach.

6.1 Semantic Role Labeling

For the past two years CoNLL has had shared tasks on SRL (Carreras and Màrquez (2004) and Carreras and Màrquez (2005)). We used the CoNLL 2005 data and evaluation script. When evaluating semantic role labeling results, it is common to present numbers on both the core arguments (i.e., excluding the modifying arguments) and all arguments. We follow this convention and present both sets of numbers. We give precision, recall and F-measure, which are based on the number of arguments correctly identified. For an argument to be correct both the span and the classification must be correct; there is no partial credit.

SRL Results – Penn Treebank Portion			
Core Args	Precision	Recall	F-measure
Greedy	79.31%	77.7%	78.50%
K-Best	80.05%	78.45%	79.24%
Sampling	80.13%	78.25%	79.18%
All Args	Precision	Recall	F-measure
Greedy	78.49%	74.77%	76.58%
K-Best	79.58%	74.90%	77.16%
Sampling	79.81%	74.85%	77.31%
SRL Results – Brown Portion			
Core Args	Precision	Recall	F-measure
Greedy	68.28%	67.72%	68.0%
K-Best	69.25%	69.02%	69.13%
Sampling	69.35%	68.93%	69.16%
All Args	Precision	Recall	F-measure
Greedy	66.6%	60.45%	63.38%
K-Best	68.82%	61.03%	64.69%
Sampling	68.6%	61.11%	64.64%

Table 1: Results for semantic role labeling task. The sampled numbers are averaged over several runs, as discussed.

To generate sampled parses, we used the Stanford parser (Klein and Manning, 2003). The CoNLL data comes with parses from Charniak's parser (Charniak, 2000), so we had to re-parse the data and retrain the SRL system on these new parses, resulting in a lower baseline than previously presented work. We choose to use Stanford's parser because of the ease with which we could modify it to generate samples. Unfortunately, its performance is slightly below that of the other parsers.

The CoNLL data has two separate test sets; the first is section 23 of the Penn Treebank (PTB), and the second is "fresh sentences" taken from the Brown corpus. For full results, please see Table 1. On the Penn Treebank portion we saw an absolute F-score improvement of 0.7% on both core and all arguments. On the Brown portion of the test set we saw an improvement of 1.25% on core and 1.16% on all arguments. In this context, a gain of over 1% is quite large: for instance, the scores for the top 4 systems on the Brown data at CoNLL 2005 were within 1% of each other. For both portions, we generated 50 samples, and did this 4 times, averaging the results. We most likely saw better performance on the Brown portion than the PTB portion because the parser was trained on the Penn Treebank training data, so the most likely parses will be of higher quality for the PTB portion of the test data than for the Brown portion. We also

[5] They report their results on the first PASCAL dataset, and use only the development set from the first challenge for learning weights. When we test on the data from the second challenge, we use all data from the first challenge and the development data from the second challenge to learn these weights.

RTE Results		
	Accuracy	Average Precision
Greedy	59.13%	59.91%
Sampling	60.88%	61.99%

Table 2: Results for recognizing textual entailment. The sampled numbers are averaged over several runs, as discussed.

ran the pipeline using a 50-best list, and found the two results to be comparable.

6.2 Textual Entailment

For the second PASCAL RTE challenge, two different types of performance measures were used to evaluate labels and confidence of the labels for the text-hypothesis pairs. The first measure is accuracy – the percentage of correct judgments. The second measure is *average precision*. Responses are sorted based on entailment confidence and then average precision is calculated by the following equation:

$$\frac{1}{R}\sum_{i=1}^{n} E(i)\frac{\text{\# correct up to pair } i}{i} \qquad (5)$$

where n is the size of the test set, R is the number of positive (entailed) examples, $E(i)$ is an indicator function whose value is 1 if the ith pair is entailed, and the is are sorted based on the entailment confidence. The intention of this measure is to evaluate how well calibrated a system is. Systems which are more confident in their correct answers and less confident in their incorrect answers will perform better on this measure.

Our results are presented in Table 2. We generated 25 samples for each run, and repeated the process 7 times, averaging over runs. Accuracy was improved by 1.5% and average precision by 2%. It does not come as a surprise that the average precision improvement was larger than the accuracy improvement, because our model explicitly estimates its own degree of confidence by estimating the posterior probability of the class label.

7 Conclusions and Future Work

We have presented a method for handling language processing pipelines in which later stages of processing are conditioned on the results of earlier stages. Currently, common practice is to take the best labeling at each point in a linguistic analysis pipeline, but this method ignores information about alternate labelings and their likelihoods. Our approach uses all of the information available, and has the added advantage of being extremely simple to implement. By modifying your subtasks to generate samples instead of the most likely labeling, our method can be used with very little additional overhead. And, as we have shown, such modifications are usually simple to make; further, with only a "small" (polynomial) number of samples k, under mild assumptions the classification error obtained by the sampling approximation approaches that of exact inference. (Ng and Jordan, 2001) In contrast, an algorithm that keeps track only of the k-best list enjoys no such theoretical guarantee, and can require an exponentially large value for k to approach comparable error. We also note that in practice, k-best lists are often more complicated to implement and more computationally expensive (e.g. the complexity of generating k sample parses or CRF outputs is substantially lower than that of generating the k best parse derivations or CRF outputs).

The major contribution of this work is not specific to semantic role labeling or recognizing textual entailment. We are proposing a general method to deal with all multi-stage algorithms. It is common to build systems using many different software packages, often from other groups, and to string together the 1-best outputs. If, instead, all NLP researchers wrote packages which can generate samples from the posterior, then the entire NLP community could use this method as easily as they can use the greedy methods that are common today, and which do not perform as well.

One possible direction for improvement of this work would be to move from a Bayesian network to an undirected Markov network. This is desirable because influence should be able to flow in both directions in this pipeline. For example, the semantic role labeler should be able to tell the parser that it did not like a particular parse, and this should influence the probability assigned to that parse. The main difficulty here lies in how to model this reversal of influence. The problem of using parse trees to help decide good semantic role labelings is well studied, but the problem of using semantic role labelings to influence parses is not. Furthermore, this requires building joint models over adjacent nodes, which is usually a nontrivial task. However, we feel that this approach would improve performance even more on these pipelined tasks and should be pursued.

8 Acknowledgements

We would like to thank our anonymous reviewers for their comments and suggestions. We would also like to thank Kristina Toutanova, Aria Haghighi and the Stanford RTE group for their assistance in understanding and using their code.

This paper is based on work funded in part by a Stanford School of Engineering fellowship and in part by the Defense Advanced Research Projects Agency through IBM. The content does not necessarily reflect the views of the U.S. Government, and no official endorsement should be inferred.

References

Rens Bod. 1995. The problem of computing the most probable tree in data-oriented parsing and stochastic tree grammars. In *Proceedings of EACL 1995*.

Xavier Carreras and Lluís Màrquez. 2004. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Proceedings of CoNLL 2004*.

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of CoNLL 2005*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 14th National Conference on Artificial Intelligence*.

Robert G. Cowell, A. Philip Dawid, Steffen L. Lauritzen, and David J. Spiegelhalter. 2003. *Probabilistic Networks and Expert Systems*. Springer.

Richard Crouch. 2005. Packed rewriting for mapping semantics to KR. In *Proceedings of the 6th International Workshop on Computational Semantics*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognizing textual entailment challenge. In *Proceedings of the PASCAL Challenges Workshop on Recognizing Textual Entailment*.

Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of ACL 2002*.

Joshua Goodman. 1998. *Parsing Inside-Out*. Ph.D. thesis, Harvard University.

Aria Haghighi, Kristina Toutanova, and Christopher D. Manning. 2005. A joint model for semantic role labeling. In *Proceedings of CoNLL 2005*.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies*.

Lauri Karttunen. 2000. Applications of finite-state transducers in natural-language processing. In *Proceesings of the Fifth International Conference on Implementation and Application of Automata*.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL 2003*.

Peter Koomen, Vasin Punyakanok, Dan Roth, and Wen tau Yih. 2005. Generalized inference with multiple semantic role labeling systems. In *Proceedings of CoNLL 2005*, pages 181–184.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289.

Bill MacCartney, Trond Grenager, Marie de Marneffe, Daniel Cer, and Christopher D. Manning. 2006. Learning to recognize features of valid textual entailments. In *Proceedings of NAACL-HTL 2006*.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts.

John T. Maxwell, III and Ronald M. Kaplan. 1995. A method for disjunctive constraint satisfaction. In Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen, editors, *Formal Issues in Lexical-Functional Grammar*, number 47 in CSLI Lecture Notes Series, chapter 14, pages 381–481. CSLI Publications.

Andrew Ng and Michael Jordan. 2001. Convergence rates of the voting Gibbs classifier, with application to Bayesian feature selection. In *Proceedings of the Eighteenth International Conference on Machine Learning*.

Charles Sutton and Andrew McCallum. 2005. Joint parsing and semantic role labeling. In *Proceedings of CoNLL 2005*, pages 225–228.

Kristina Toutanova, Aria Haghighi, and Christopher D. Manning. 2005. Joint learning improves semantic role labeling. In *Proceedings of ACL 2005*.

Kristina Toutanova. 2005. *Effective statistical models for syntactic and semantic disambiguation*. Ph.D. thesis, Stanford University.

Ben Wellner, Andrew McCallum, Fuchun Peng, and Michael Hay. 2004. An integrated, conditional model of information extraction and coreference with application to citation matching. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence*.