Fast Syntactic Analysis for Statistical Language Modeling via Substructure Sharing and Uptraining

Ariya Rastrow, Mark Dredze, Sanjeev Khudanpur
Human Language Technology Center of Excellence
Center for Language and Speech Processing, Johns Hopkins University
Baltimore, MD USA
{ariya,mdredze,khudanpur}@jhu.edu

Abstract

Long-span features, such as syntax, can improve language models for tasks such as speech recognition and machine translation. However, these language models can be difficult to use in practice because of the time required to generate features for rescoring a large hypothesis set. In this work, we propose substructure sharing, which saves duplicate work in processing hypothesis sets with redundant hypothesis structures. We apply substructure sharing to a dependency parser and part of speech tagger to obtain significant speedups, and further improve the accuracy of these tools through up-training. When using these improved tools in a language model for speech recognition, we obtain significant speed improvements with both N-best and hill climbing rescoring, and show that up-training leads to WER reduction.

1 Introduction

Language models (LM) are crucial components in tasks that require the generation of coherent natural language text, such as automatic speech recognition (ASR) and machine translation (MT). While traditional LMs use word n-grams, where the $n-1$ previous words predict the next word, newer models integrate long-span information in making decisions. For example, incorporating long-distance dependencies and syntactic structure can help the LM better predict words by complementing the predictive power of n-grams (Chelba and Jelinek, 2000; Collins et al., 2005; Filimonov and Harper, 2009; Kuo et al., 2009).

The long-distance dependencies can be modeled in either a *generative* or a *discriminative* framework. Discriminative models, which directly distinguish correct from incorrect hypothesis, are particularly attractive because they allow the inclusion of arbitrary features (Kuo et al., 2002; Roark et al., 2007; Collins et al., 2005); these models with syntactic information have obtained state of the art results.

However, both generative and discriminative LMs with long-span dependencies can be slow, for they often cannot work directly with lattices and require *rescoring* large N-best lists (Khudanpur and Wu, 2000; Collins et al., 2005; Kuo et al., 2009). For discriminative models, this limitation applies to training as well. Moreover, the non-local features used in rescoring are usually extracted via auxiliary tools – which in the case of syntactic features include part of speech taggers and parsers – from a set of ASR system hypotheses. Separately applying auxiliary tools to each N-best list hypothesis leads to major inefficiencies as many hypotheses differ only slightly.

Recent work on hill climbing algorithms for ASR lattice rescoring iteratively searches for a higher-scoring hypothesis in a local neighborhood of the current-best hypothesis, leading to a much more efficient algorithm in terms of the number, N, of hypotheses evaluated (Rastrow et al., 2011b); the idea also leads to a discriminative hill climbing training algorithm (Rastrow et al., 2011a). Even so, the reliance on auxiliary tools slow LM application to the point of being impractical for real time systems. While faster auxiliary tools are an option, they are usually less accurate.

In this paper, we propose a general modification to the decoders used in auxiliary tools to utilize the commonalities among the set of generated hypotheses. The key idea is to share substructure states in transition based structured prediction algorithms, i.e. algorithms where final structures are composed of a sequence of multiple individual decisions. We demonstrate our approach on a local Perceptron based part of speech tagger (Tsuruoka et al., 2011) and a shift reduce dependency parser (Sagae and Tsujii, 2007), yielding significantly faster tagging and parsing of ASR hypotheses. While these simpler structured prediction models are faster, we compensate for the model's simplicity through uptraining (Petrov et al., 2010), yielding auxiliary tools that are both fast and accurate. The result is significant speed improvements and a reduction in word error rate (WER) for both N-best list and the already fast hill climbing rescoring. The net result is arguably the first syntactic LM fast enough to be used in a *real time ASR system*.

2 Syntactic Language Models

There have been several approaches to include syntactic information in both generative and discriminative language models.

For generative LMs, the syntactic information must be part of the generative process. Structured language modeling incorporates syntactic parse trees to identify the head words in a hypothesis for modeling dependencies beyond n-grams. Chelba and Jelinek (2000) extract the two previous exposed head words at each position in a hypothesis, along with their non-terminal tags, and use them as context for computing the probability of the current position. Khudanpur and Wu (2000) exploit such syntactic head word dependencies as features in a maximum entropy framework. Kuo et al. (2009) integrate syntactic features into a neural network LM for Arabic speech recognition.

Discriminative models are more flexible since they can include arbitrary features, allowing for a wider range of long-span syntactic dependencies. Additionally, discriminative models are directly trained to resolve the acoustic confusion in the decoded hypotheses of an ASR system. This flexibility and training regime translate into better performance. Collins et al. (2005) uses the Perceptron algorithm to train a global linear discriminative model which incorporates long-span features, such as head-to-head dependencies and part of speech tags.

Our Language Model. We work with a discriminative LM with long-span dependencies. We use a global linear model with Perceptron training. We rescore the hypotheses (lattices) generated by the ASR decoder—in a framework most similar to that of Rastrow et al. (2011a).

The LM score $\mathcal{S}(\mathbf{w}, \mathbf{a})$ for each hypothesis $\mathbf{w}$ of a speech utterance with acoustic sequence $\mathbf{a}$ is based on the baseline ASR system score $b(\mathbf{w}, \mathbf{a})$ (initial n-gram LM score and the acoustic score) and α_0, the weight assigned to the baseline score.[1] The score is defined as:

$$\begin{aligned}\mathcal{S}(\mathbf{w}, \mathbf{a}) &= \alpha_0 \cdot b(\mathbf{w}, \mathbf{a}) + F(\mathbf{w}, \mathbf{s}^1, \ldots, \mathbf{s}^m) \\ &= \alpha_0 \cdot b(\mathbf{w}, \mathbf{a}) + \sum_{i=1}^{d} \alpha_i \cdot \Phi_i(\mathbf{w}, \mathbf{s}^1, \ldots, \mathbf{s}^m)\end{aligned}$$

where F is the discriminative LM's score for the hypothesis $\mathbf{w}$, and $\mathbf{s}^1, \ldots, \mathbf{s}^m$ are candidate syntactic structures associated with $\mathbf{w}$, as discussed below. Since we use a linear model, the score is a weighted linear combination of the *count* of activated features of the word sequence $\mathbf{w}$ and its associated structures: $\Phi_i(\mathbf{w}, \mathbf{s}^1, \ldots, \mathbf{s}^m)$. Perceptron training learns the parameters $\boldsymbol{\alpha}$. The baseline score $b(\mathbf{w}, \mathbf{a})$ can be a feature, yielding the dot product notation: $\mathcal{S}(\mathbf{w}, \mathbf{a}) = \langle \boldsymbol{\alpha}, \Phi(\mathbf{a}, \mathbf{w}, \mathbf{s}^1, \ldots, \mathbf{s}^m) \rangle$ Our LM uses features from the dependency tree and part of speech (POS) tag sequence. We use the method described in Kuo et al. (2009) to identify the two previous exposed head words, h_{-2}, h_{-1}, at each position i in the input hypothesis and include the following syntactic based features into our LM:

1. $(h_{-2}.\mathrm{w} \circ h_{-1}.\mathrm{w} \circ w_i)$, $(h_{-1}.\mathrm{w} \circ w_i)$, (w_i)
2. $(h_{-2}.\mathrm{t} \circ h_{-1}.\mathrm{t} \circ t_i)$, $(h_{-1}.\mathrm{t} \circ t_i)$, (t_i) , $(t_i w_i)$

where $h.\mathrm{w}$ and $h.\mathrm{t}$ denote the word identity and the POS tag of the corresponding exposed head word.

2.1 Hill Climbing Rescoring

We adopt the so called hill climbing framework of Rastrow et al. (2011b) to improve both training and rescoring time as much as possible by reducing the

[1]We tune α_0 on development data (Collins et al., 2005).

number N of explored hypotheses. We summarize it below for completeness.

Given a speech utterance's lattice $\mathcal{L}$ from a first pass ASR decoder, the neighborhood $\mathcal{N}(\mathbf{w}, i)$ of a hypothesis $\mathbf{w} = w_1 w_2 \ldots w_n$ at position i is defined as the set of all paths in the lattice that may be obtained by editing w_i: deleting it, substituting it, or inserting a word to its left. In other words, it is the "distance-1-at-position i" neighborhood of $\mathbf{w}$. Given a position i in a word sequence $\mathbf{w}$, all hypotheses in $\mathcal{N}(\mathbf{w}, i)$ are rescored using the long-span model and the hypothesis $\hat{\mathbf{w}}'(i)$ with the highest score becomes the new $\mathbf{w}$. The process is repeated with a new position – scanned left to right – until $\mathbf{w} = \hat{\mathbf{w}}'(1) = \ldots = \hat{\mathbf{w}}'(n)$, i.e. when $\mathbf{w}$ itself is the highest scoring hypothesis in all its 1-neighborhoods, and can not be furthered improved using the model. Incorporating this into training yields a discriminative hill climbing algorithm (Rastrow et al., 2011a).

3 Incorporating Syntactic Structures

Long-span models – generative or discriminative, N-best or hill climbing – rely on auxiliary tools, such as a POS tagger or a parser, for extracting features for each hypothesis during rescoring, and during training for discriminative models. The top-m candidate structures associated with the i^{th} hypothesis, which we denote as $\mathbf{s}_i^1, \ldots, \mathbf{s}_i^m$, are generated by these tools and used to score the hypothesis: $F(\mathbf{w}_i, \mathbf{s}_i^1, \ldots, \mathbf{s}_i^m)$. For example, $\mathbf{s}_i^j$ can be a part of speech tag or a syntactic dependency. We formally define this sequential processing as:

$$\begin{array}{l} \mathbf{w}_1 \xrightarrow{\text{tool(s)}} \mathbf{s}_1^1, \ldots, \mathbf{s}_1^m \xrightarrow{\text{LM}} F(\mathbf{w}_1, \mathbf{s}_1^1, \ldots, \mathbf{s}_1^m) \\ \mathbf{w}_2 \xrightarrow{\text{tool(s)}} \mathbf{s}_2^1, \ldots, \mathbf{s}_2^m \xrightarrow{\text{LM}} F(\mathbf{w}_2, \mathbf{s}_2^1, \ldots, \mathbf{s}_2^m) \\ \vdots \\ \mathbf{w}_k \xrightarrow{\text{tool(s)}} \mathbf{s}_k^1, \ldots, \mathbf{s}_k^m \xrightarrow{\text{LM}} F(\mathbf{w}_k, \mathbf{s}_k^1, \ldots, \mathbf{s}_k^m) \end{array}$$

Here, $\{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ represents a set of ASR output hypotheses that need to be rescored. For each hypothesis, we apply an external tool (e.g. parser) to generate associated structures $\mathbf{s}_i^1, \ldots, \mathbf{s}_i^m$ (e.g. dependencies.) These are then passed to the language model along with the word sequence for scoring.

3.1 Substructure Sharing

While long-span LMs have been empirically shown to improve WER over n-gram LMs, the computational burden prohibits long-span LMs in practice, particularly in real-time systems. A major complexity factor is due to processing 100s or 1000s of hypotheses for each speech utterance, even during hill climbing, each of which must be POS tagged and parsed. However, the candidate hypotheses of an utterance share equivalent substructures, especially in hill climbing methods due to the locality present in the neighborhood generation. Figure 1 demonstrates such repetition in an N-best list (N=10) and a hill climbing neighborhood hypothesis set for a speech utterance from broadcast news. For example, the word "ENDORSE" occurs within the same local context in all hypotheses and should receive the same part of speech tag in each case. Processing each hypothesis separately wastes time.

We propose a general algorithmic approach to reduce the complexity of processing a hypothesis set by sharing common substructures among the hypotheses. Critically, unlike many lattice parsing algorithms, our approach is general and produces exact output. We first present our approach and then demonstrate its generality by applying it to a dependency parser and part of speech tagger.

We work with structured prediction models that produce output from a series of local decisions: a transition model. We begin in initial state π_0 and terminate in a possible final state π_f. All states along the way are chosen from the possible states Π. A transition (or action) $\omega \in \Omega$ advances the decoder from state to state, where the transition ω_i changes the state from π_i to π_{i+1}. The sequence of states $\{\pi_0 \ldots \pi_i, \pi_{i+1} \ldots \pi_f\}$ can be mapped to an output (the model's prediction.) The choice of action ω is given by a learning algorithm, such as a maximum-entropy classifier, support vector machine or Perceptron, trained on labeled data. Given the previous k actions up to π_i, the classifier $g : \Pi \times \Omega^k \rightarrow \mathbb{R}^{|\Omega|}$ assigns a score to each possible action, which we can interpret as a probability: $p_g(\omega_i | \pi_i, \omega_{i-1}\omega_{i-2} \ldots \omega_{i-k})$. These actions are applied to transition to new states π_{i+1}. We note that state definitions can encode the k previous actions, which simplifies the probability to $p_g(\omega_i | \pi_i)$. The

N-best list		Hill climbing neighborhood	
(1)	AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE		
(2)	TO AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE		
(3)	AL GORE HAS PROMISE THAT HE WOULD ENDORSE A CANDIDATE		
(4)	SO AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE	(1)	YEAH FIFTY CENT GALLON NOMINATION WHICH WAS GREAT
(5)	IT'S AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE	(2)	YEAH FIFTY CENT A GALLON NOMINATION WHICH WAS GREAT
(6)	AL GORE HAS PROMISED HE WOULD ENDORSE A CANDIDATE	(3)	YEAH FIFTY CENT GOT A NOMINATION WHICH WAS GREAT
(7)	AL GORE HAS PROMISED THAT HE WOULD ENDORSE THE CANDIDATE		
(8)	SAID AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE		
(9)	AL GORE HAS PROMISED THAT HE WOULD ENDORSE A CANDIDATE FOR		
(10)	AL GORE HIS PROMISE THAT HE WOULD ENDORSE A CANDIDATE		

Figure 1: Example of repeated substructures in candidate hypotheses.

score of the new state is then

$$p(\pi_{i+1}) = p_g(\omega_i|\pi_i) \cdot p(\pi_i) \tag{1}$$

Classification decisions require a feature representation of π_i, which is provided by feature functions $\mathbf{f} : \Pi \rightarrow \mathcal{Y}$, that map states to features. Features are conjoined with actions for multi-class classification, so $p_g(\omega_i|\pi_i) = p_g(\mathbf{f}(\pi) \circ \omega_i)$, where $\circ$ is a conjunction operation. In this way, states can be summarized by features.

Equivalent states are defined as two states π and π' with an identical feature representation:

$$\pi \equiv \pi' \quad \text{iff} \quad \mathbf{f}(\pi) = \mathbf{f}(\pi')$$

If two states are equivalent, then g imposes the same distribution over actions. We can benefit from this substructure redundancy, both within and between hypotheses, by saving these distributions in memory, sharing a distribution computed just once across equivalent states. A similar idea of equivalent states is used by Huang and Sagae (2010), except they use equivalence to facilitate dynamic programming for shift-reduce parsing, whereas we generalize it for improving the processing time of similar hypotheses in general models. Following Huang and Sagae, we define *kernel features* as the smallest set of atomic features $\tilde{\mathbf{f}}(\pi)$ such that,

$$\tilde{\mathbf{f}}(\pi) = \tilde{\mathbf{f}}(\pi') \quad \Rightarrow \quad \pi \equiv \pi'. \tag{2}$$

Equivalent distributions are stored in a *hash table* $H : \Pi \rightarrow \Omega \times \mathbb{R}$; the hash keys are the states and the values are distributions[2] over actions: $\{\omega, p_g(\omega|\pi)\}$. H caches equivalent states in a hypothesis set and resets for each new utterance. For each state, we first check H for equivalent states before computing the action distribution; each cache hit reduces decoding time. Distributing hypotheses $\mathbf{w}_i$ across different CPU threads is another way to obtain speedups, and we can still benefit from substructure sharing by storing H in shared memory.

We use $h(\pi) = \sum_{i=1}^{|\tilde{\mathbf{f}}(\pi)|} \text{int}(\tilde{f}_i(\pi))$ as the hash function, where $\text{int}(\tilde{f}_i(\pi))$ is an integer mapping of the i^{th} kernel feature. For integer typed features the mapping is trivial, for string typed features (e.g. a POS tag identity) we use a mapping of the corresponding vocabulary to integers. We empirically found that this hash function is very effective and yielded very few collisions.

To apply substructure sharing to a transition based model, we need only define the set of states Π (including π_0 and π_f), actions Ω and kernel feature functions $\tilde{\mathbf{f}}$. The resulting speedup depends on the amount of substructure duplication among the hypotheses, which we will show is significant for ASR lattice rescoring. Note that our algorithm is not an approximation; we obtain the same output $\{\mathbf{s}_i^j\}$ as we would without any sharing. We now apply this algorithm to dependency parsing and POS tagging.

3.2 Dependency Parsing

We use the best-first probabilistic shift-reduce dependency parser of Sagae and Tsujii (2007), a transition-based parser (Kübler et al., 2009) with a MaxEnt classifier. Dependency trees are built by processing the words left-to-right and the classifier assigns a distribution over the actions at each step. States are defined as $\pi = \{S, Q\}$: S is a stack of

[2]For pure greedy search (deterministic search) we need only retain the best action, since the distribution is only used in probabilistic search, such as beam search or best-first algorithms.

Kernel features $\tilde{\mathbf{f}}(\pi)$ for state $\pi = \{S, Q\}$ $S = s_0, s_1, \ldots$ & $Q = q_0, q_1, \ldots$					
(1)	s_0.w	s_0.t	$s_0.r$	(5)	t_{s_0-1}
		s_0.lch.t	s_0.lch.r		t_{s_1+1}
		s_0.rch.t	s_0.rch.r		
(2)	s_1.w	s_1.t	$s_1.r$	(6)	dist(s_0, s_1)
		s_1.lch.t	s_1.lch.r		dist(q_0, s_0)
		s_1.rch.t	s_1.rch.r		
(3)	s_2.w	s_2.t	$s_2.r$		
(4)	q_0.w	q_0.t		(7)	s_0.nch
	q_1.w	q_1.t			s_1.nch
	q_2.w				

Table 1: Kernel features for defining parser states. s_i.w denotes the head-word in a subtree and t its POS tag. s_i.lch and s_i.rch are the leftmost and rightmost children of a subtree. $s_i.r$ is the dependency label that relates a subtree head-word to its dependent. s_i.nch is the number of children of a subtree. q_i.w and q_i.t are the word and its POS tag in the queue. dist(s_0,s_1) is the linear distance between the head-words of s_0 and s_1.

Algorithm 1 Best-first shift-reduce dependency parsing

```
$\mathbf{w} \leftarrow$ input hypothesis
$S_0 = \emptyset, Q_0 = \mathbf{w}, p_0 = 1$
$\pi_0 \leftarrow \{S_0, Q_0, p_0\}$                                  [initial state]
$H \leftarrow$Hash table ($\Pi \rightarrow \Omega \times \mathbb{R}$)
Heap$\leftarrow$ Heap for prioritizing states and performing best-first search
Heap.push($\pi_0$)                                                   [initialize the heap]

**while** Heap $\neq \emptyset$ **do**
    $\pi_{\text{current}} \leftarrow$Heap.pop()                         [the best state so far]
    **if** $\pi_{\text{current}} = \pi_f$                                    [if final state]
      return $\pi_{\text{current}}$                                     [terminate if final state]
    **else if** $H$.find($\pi_{\text{current}}$)
      ActList $\leftarrow H[\pi_{\text{current}}]$                 [retrieve action list from the hash table]
    **else**                                                         [need to construct action list]
      **for all** $\omega \in \Omega$                                         [for all actions]
        $p_\omega \leftarrow p_g(\omega|\pi_{\text{current}})$                               [action score]
        ActList.insert($\{\omega, p_\omega\}$)
      $H$.insert($\pi_{\text{current}}$, ActList)                [Store the action list into hash table]
    **end if**
    **for all** $\{\omega, p_\omega\} \in$ ActList                                [compute new states]
      $\pi_{\text{new}} \leftarrow \pi_{\text{current}} \times \omega$
      Heap.push($\pi_{\text{new}}$)                                       [push to the heap]
**end while**
```

subtrees $s_0, s_1, \ldots$ (s_0 is the top tree) and Q are words in the input word sequence. The initial state is $\pi_0 = \{\emptyset, \{w_0, w_1, \ldots\}\}$, and final states occur when Q is empty and S contains a single tree (the output).

Ω is determined by the set of dependency labels $r \in \mathcal{R}$ and one of three transition types:

- *Shift*: remove the head of Q (w_j) and place it on the top of S as a singleton tree (only w_j.)
- *Reduce-Left$_r$*: replace the top two trees in S (s_0 and s_1) with a tree formed by making the root of s_1 a dependent of the root of s_0 with label r.
- *Reduce-Right$_r$*: same as *Reduce-Left$_r$* except reverses s_0 and s_1.

Table 1 shows the kernel features used in our dependency parser. See Sagae and Tsujii (2007) for a complete list of features.

Goldberg and Elhadad (2010) observed that parsing time is dominated by feature extraction and score calculation. Substructure sharing reduces these steps for equivalent states, which are persistent throughout a candidate set. Note that there are far fewer kernel features than total features, hence the hash function calculation is very fast.

We summarize substructure sharing for dependency parsing in Algorithm 1. We extend the definition of states to be $\{S, Q, p\}$ where p denotes the score of the state: the probability of the action sequence that resulted in the current state. Also, following Sagae and Tsujii (2007) a heap is used to maintain states prioritized by their scores, for applying the best-first strategy. For each step, a state from the top of the heap is considered and all actions (and scores) are either retrieved from H or computed using g.[3] We use $\pi_{\text{new}} \leftarrow \pi_{\text{current}} \times \omega$ to denote the operation of extending a state by an action $\omega \in \Omega$[4].

3.3 Part of Speech Tagging

We use the part of speech (POS) tagger of Tsuruoka et al. (2011), a transition based model with a Perceptron and a lookahead heuristic process. The tagger processes $\mathbf{w}$ left to right. States are defined as $\pi_i = \{c_i, \mathbf{w}\}$: a sequence of assigned tags up to w_i ($c_i = t_1 t_2 \ldots t_{i-1}$) and the word sequence $\mathbf{w}$. Ω is defined simply as the set of possible POS tags ($\mathcal{T}$) that can be applied. The final state is reached once all the positions are tagged. For $\mathbf{f}$ we use the features of Tsuruoka et al. (2011). The kernel features are $\tilde{\mathbf{f}}(\pi_i) = \{t_{i-2}, t_{i-1}, w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}\}$. While the tagger extracts prefix and suffix features, it suffices to look at w_i for determining state equivalence. The tagger is deterministic (greedy) in that it only considers the best tag at each step, so we do not store scores. However, this tagger uses a depth-

[3] Sagae and Tsujii (2007) use a beam strategy to increase speed. Search space pruning is achieved by filtering heap states for probability greater than $\frac{1}{b}$ the probability of the most likely state in the heap with the same number of actions. We use $b = 100$ for our experiments.

[4] We note that while we have demonstrated substructure sharing for **dependency** parsing, the same improvements can be made to a shift-reduce constituent parser (Sagae and Lavie, 2006).

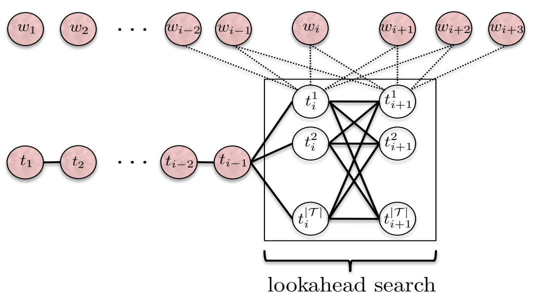


Figure 2: POS tagger with lookahead search of d=1. At w_i the search considers the current state and next state.

first search lookahead procedure to select the best action at each step, which considers future decisions up to depth d[5]. An example for $d = 1$ is shown in Figure 2. Using $d = 1$ for the lookahead search strategy, we modify the kernel features since the decision for w_i is affected by the state π_{i+1}. The kernel features in position i should be $\tilde{\mathbf{f}}(\pi_i) \cup \tilde{\mathbf{f}}(\pi_{i+1})$:

$$\tilde{\mathbf{f}}(\pi_i) = \{t_{i-2}, t_{i-1}, w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, w_{i+3}\}$$

4 Up-Training

While we have fast decoding algorithms for the parsing and tagging, the simpler underlying models can lead to worse performance. Using more complex models with higher accuracy is impractical because they are slow. Instead, we seek to improve the accuracy of our fast tools.

To achieve this goal we use up-training, in which a more complex model is used to improve the accuracy of a simpler model. We are given two models, M_1 and M_2, as well as a large collection of unlabeled text. Model M_1 is slow but very accurate while M_2 is fast but obtains lower accuracy. Up-training applies M_1 to tag the unlabeled data, which is then used as training data for M_2. Like self-training, a model is retrained on automatic output, but here the output comes form a more accurate model. Petrov et al. (2010) used up-training as a domain adaptation technique: a constituent parser – which is more robust to domain changes – was used to label a new domain, and a fast dependency parser was trained on the automatically labeled data. We use a similar idea where our goal is to recover the accuracy lost from using simpler models. Note that while up-training uses two models, it differs from co-training since we care about improving only one model (M_2). Additionally, the models can vary in different ways. For example, they could be the same algorithm with different pruning methods, which can lead to faster but less accurate models.

We apply up-training to improve the accuracy of both our fast POS tagger and dependency parser. We parse a large corpus of text with a very accurate but very slow constituent parser and use the resulting data to up-train our tools. We will demonstrate empirically that up-training improves these fast models to yield better WER results.

5 Related Work

The idea of efficiently processing a hypothesis set is similar to "lattice-parsing", in which a parser consider an entire lattice at once (Hall, 2005; Cheppalier et al., 1999). These methods typically constrain the parsing space using heuristics, which are often model specific. In other words, they search in the joint space of word sequences present in the lattice and their syntactic analyses; they are not guaranteed to produce a syntactic analysis for all hypotheses. In contrast, substructure sharing is a general purpose method that we have applied to two different algorithms. The output is identical to processing each hypothesis separately and output is generated for each hypothesis. Hall (Hall, 2005) uses a lattice parsing strategy which aims to compute the marginal probabilities of all word sequences in the lattice by summing over syntactic analyses of each word sequence. The parser sums over multiple parses of a word sequence implicitly. The lattice parser therefore, is itself a language model. In contrast, our tools are completely separated from the ASR system, which allows the system to create whatever features are needed. This independence means our tools are useful for other tasks, such as machine translation. These differences make substructure sharing a more attractive option for efficient algorithms.

While Huang and Sagae (2010) use the notion of "equivalent states", they do so for dynamic programming in a shift-reduce parser to broaden the search space. In contrast, we use the idea to identify sub-

[5] Tsuruoka et al. (2011) shows that the lookahead search improves the performance of the local "history-based" models for different NLP tasks

structures across inputs, where our goal is efficient parsing in general. Additionally, we extend the definition of equivalent states to general transition based structured prediction models, and demonstrate applications beyond parsing as well as the novel setting of hypothesis set parsing.

6 Experiments

Our ASR system is based on the 2007 IBM Speech transcription system for the GALE Distillation Go/No-go Evaluation (Chen et al., 2006) with state of the art discriminative acoustic models. See Table 2 for a data summary. We use a modified Kneser-Ney (KN) backoff 4-gram baseline LM. Word-lattices for discriminative training and rescoring come from this baseline ASR system.[6] The long-span discriminative LM's baseline feature weight (α_0) is tuned on dev data and hill climbing (Rastrow et al., 2011a) is used for training and rescoring. The dependency parser and POS tagger are trained on supervised data and up-trained on data labeled by the CKY-style bottom-up constituent parser of Huang et al. (2010), a state of the art broadcast news (BN) parser, with phrase structures converted to labeled dependencies by the Stanford converter.

While accurate, the parser has a huge grammar (32GB) from using products of latent variable grammars and requires $O(l^3)$ time to parse a sentence of length l. Therefore, we could not use the constituent parser for ASR rescoring since utterances can be very long, although the shorter up-training text data was not a problem.[7] We evaluate both unlabeled (UAS) and labeled dependency accuracy (LAS).

6.1 Results

Before we demonstrate the speed of our models, we show that up-training can produce accurate and fast models. Figure 3 shows improvements to parser accuracy through up-training for different amount of (randomly selected) data, where the last column indicates constituent parser score (91.4% UAS). We use the POS tagger to generate tags for dependency training to match the test setting. While there is a large difference between the constituent and dependency parser without up-training (91.4%

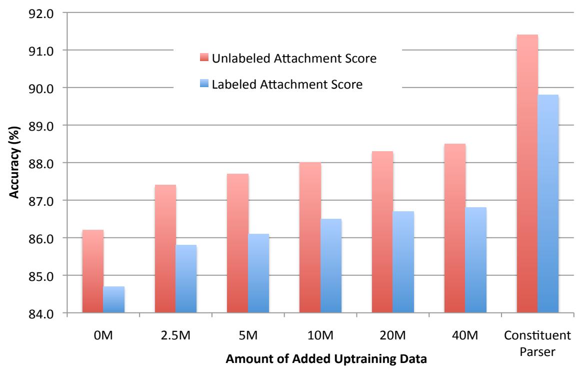


Figure 3: Up-training results for dependency parsing for varying amounts of data (number of words.) The first column is the dependency parser with supervised training only and the last column is the constituent parser (after converting to dependency trees.)

vs. 86.2% UAS), up-training can cut the difference by 44% to 88.5%, and improvements saturate around 40m words (about 2m sentences.)[8] The dependency parser remains much smaller and faster; the up-trained dependency model is 700MB with 6m features compared with 32GB for constituency model. Up-training improves the POS tagger's accuracy from 95.9% to 97%, when trained on the POS tags produced by the constituent parser, which has a tagging accuracy of 97.2% on BN.

We train the syntactic discriminative LM, with head-word and POS tag features, using the faster parser and tagger and then rescore the ASR hypotheses. Table 3 shows the decoding speedups as well as the WER reductions compared to the baseline LM. Note that up-training improvements lead to WER reductions. Detailed speedups on substructure sharing are shown in Table 4; the POS tagger achieves a 5.3 times speedup, and the parser a 5.7 speedup without changing the output. We also observed speedups during training (not shown due to space.)

The above results are for the already fast hill climbing decoding, but substructure sharing can also be used for N-best list rescoring. Figure 4 (logarithmic scale) illustrates the time for the parser and tagger to process N-best lists of varying size, with more substantial speedups for larger lists. For example, for N=100 (a typical setting) the parsing time re-

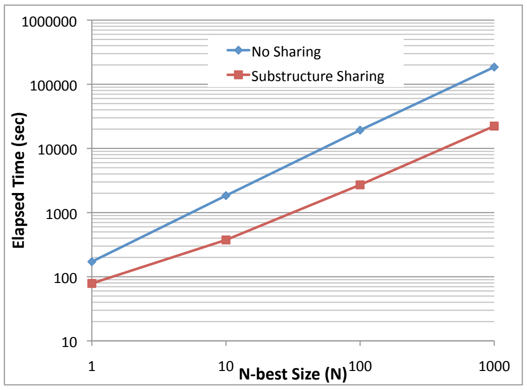
[6]For training a 3-gram LM is used to increase confusions.

[7]Speech utterances are longer as they are not as effectively sentence segmented as text.

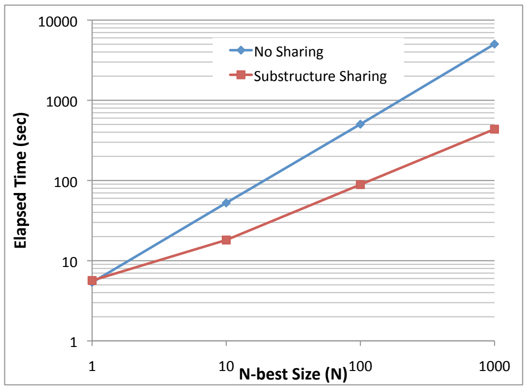
[8]Better performance is due to the exact CKY-style – compared with best-first and beam– search and that the constituent parser uses the product of huge self-trained grammars.

Usage	Data	Size
Acoustic model training	Hub4 acoustic train	153k uttr, 400 hrs
Baseline LM training: modified KN 4-gram	TDT4 closed captions+EARS BN03 closed caption	193m words
Disc. LM training: long-span w/hill climbing	Hub4 (length <50)	115k uttr, 2.6m words
Baseline feature (α_0) tuning	dev04f BN data	2.5 hrs
Supervised training: dep. parser, POS tagger	Ontonotes BN treebank+ WSJ Penn treebank	1.3m words, 59k sent.
Supervised training: constituent parser	Ontonotes BN treebank + WSJ Penn treebank	1.3m words, 59k sent.
Up-training: dependency parser, POS tagger	TDT4 closed captions+EARS BN03 closed caption	193m words available
Evaluation: up-training	BN treebank test (following Huang et al. (2010))	20k words, 1.1k sent.
Evaluation: ASR transcription	rt04 BN evaluation	4 hrs, 45k words

Table 2: A summary of the data for training and evaluation. The Ontonotes corpus is from Weischedel et al. (2008).



(a)



(b)

Figure 4: Elapsed time for (a) parsing and (b) POS tagging the N-best lists with and without substructure sharing.

		Substr. Share (sec)	
LM	WER	No	Yes
Baseline 4-gram	15.1	-	-
Syntactic LM	14.8	8,658	**1,648**
+ up-train	**14.6**		

Table 3: Speedups and WER for hill climbing rescoring. Substructure sharing yields a 5.3 times speedup. The times for with and without up-training are nearly identical, so we include only one set for clarity. Time spent is dominated by the parser, so the faster parser accounts for much of the overall speedup. Timing information includes neighborhood generation and LM rescoring, so it is more than the sum of the times in Table 4.

	Substr. Share		Speedup
	No	Yes	
Parser	8,237.2	**1,439.5**	5.7
POS tagger	213.3	**40.1**	5.3

Table 4: Time in seconds for the parser and POS tagger to process hypotheses during hill climbing rescoring.

duces from about 20,000 seconds to 2,700 seconds, about 7.4 times as fast.

7 Conclusion

The computational complexity of accurate syntactic processing can make structured language models impractical for applications such as ASR that require scoring hundreds of hypotheses per input. We have presented substructure sharing, a general framework that greatly improves the speed of syntactic tools that process candidate hypotheses. Furthermore, we achieve improved performance through up-training. The result is a large speedup in rescoring time, even on top of the already fast hill climbing framework, and reductions in WER from up-training. Our results make long-span syntactic LMs practical for real-time ASR, and can potentially impact machine translation decoding as well.

Acknowledgments

Thanks to Kenji Sagae for sharing his shift-reduce dependency parser and the anonymous reviewers for helpful comments.

References

C. Chelba and F. Jelinek. 2000. Structured language modeling. *Computer Speech and Language*, 14(4):283–332.

S. Chen, B. Kingsbury, L. Mangu, D. Povey, G. Saon, H. Soltau, and G. Zweig. 2006. Advances in speech transcription at IBM under the DARPA EARS program. *IEEE Transactions on Audio, Speech and Language Processing*, pages 1596–1608.

J. Cheppalier, M. Rajman, R. Aragues, and A. Rozenknop. 1999. Lattice parsing for speech recognition. In *Sixth Conference sur le Traitement Automatique du Langage Naturel (TANL'99)*.

M Collins, B Roark, and M Saraclar. 2005. Discriminative syntactic language modeling for speech recognition. In *ACL*.

Denis Filimonov and Mary Harper. 2009. A joint language model with fine-grain syntactic tags. In *EMNLP*.

Yoav Goldberg and Michael Elhadad. 2010. An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In *Proc. HLT-NAACL*, number June, pages 742–750.

Keith B Hall. 2005. *Best-first word-lattice parsing: techniques for integrated syntactic language modeling*. Ph.D. thesis, Brown University.

L. Huang and K. Sagae. 2010. Dynamic Programming for Linear-Time Incremental Parsing. In *Proceedings of ACL*.

Zhongqiang Huang, Mary Harper, and Slav Petrov. 2010. Self-training with Products of Latent Variable Grammars. In *Proc. EMNLP*, number October, pages 12–22.

S. Khudanpur and J. Wu. 2000. Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling. *Computer Speech and Language*, pages 355–372.

S. Kübler, R. McDonald, and J. Nivre. 2009. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 2(1):1–127.

Hong-Kwang Jeff Kuo, Eric Fosler-Lussier, Hui Jiang, and Chin-Hui Lee. 2002. Discriminative training of language models for speech recognition. In *ICASSP*.

H. K. J. Kuo, L. Mangu, A. Emami, I. Zitouni, and L. Young-Suk. 2009. Syntactic features for Arabic speech recognition. In *Proc. ASRU*.

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 705–713, Cambridge, MA, October. Association for Computational Linguistics.

Ariya Rastrow, Mark Dredze, and Sanjeev Khudanpur. 2011a. Efficient discrimnative training of long-span language models. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*.

Ariya Rastrow, Markus Dreyer, Abhinav Sethy, Sanjeev Khudanpur, Bhuvana Ramabhadran, and Mark Dredze. 2011b. Hill climbing on speech lattices : A new rescoring framework. In *ICASSP*.

Brian Roark, Murat Saraclar, and Michael Collins. 2007. Discriminative n-gram language modeling. *Computer Speech & Language*, 21(2).

K. Sagae and A. Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proc. ACL*, pages 691–698. Association for Computational Linguistics.

K. Sagae and J. Tsujii. 2007. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proc. EMNLP-CoNLL*, volume 7, pages 1044–1050.

Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Kazama. 2011. Learning with Lookahead : Can History-Based Models Rival Globally Optimized Models ? In *Proc. CoNLL*, number June, pages 238–246.

Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Martha Palmer, Nianwen Xue, Mitchell Marcus, Ann Taylor, Craig Greenberg, Eduard Hovy, Robert Belvin, and Ann Houston, 2008. *OntoNotes Release 2.0*. Linguistic Data Consortium, Philadelphia.