Word Lattice Reranking for Chinese Word Segmentation and Part-of-Speech Tagging

Wenbin Jiang [†‡] **Haitao Mi** [†‡] **Qun Liu** [†]

[†]Key Lab. of Intelligent Information Processing
Institute of Computing Technology
Chinese Academy of Sciences
P.O. Box 2704, Beijing 100190, China
[‡]Graduate University of Chinese Academy of Sciences
Beijing, 100049, China
{jiangwenbin,htmi,liuqun}@ict.ac.cn

Abstract

In this paper, we describe a new reranking strategy named word lattice reranking, for the task of joint Chinese word segmentation and part-of-speech (POS) tagging. As a derivation of the forest reranking for parsing (Huang, 2008), this strategy reranks on the pruned word lattice, which potentially contains much more candidates while using less storage, compared with the traditional n-best list reranking. With a perceptron classifier trained with local features as the baseline, word lattice reranking performs reranking with non-local features that can't be easily incorporated into the perceptron baseline. Experimental results show that, this strategy achieves improvement on both segmentation and POS tagging, above the perceptron baseline and the n-best list reranking.

1 Introduction

Recent work for Chinese word segmentation and POS tagging pays much attention to discriminative methods, such as Maximum Entropy Model (ME) (Ratnaparkhi and Adwait, 1996), Conditional Random Fields (CRFs) (Lafferty et al., 2001), perceptron training algorithm (Collins, 2002), etc. Compared to generative ones such as Hidden Markov Model (HMM) (Rabiner, 1989; Fine et al., 1998), discriminative models have the advantage of flexibility in representing features, and usually obtains almost perfect accuracy in two tasks.

Originated by Xue and Shen (2003), the typical approach of discriminative models conducts segmentation in a classification style, by assigning each character a positional tag indicating its relative position in the word. If we extend these positional tags to include POS information, segmentation and POS tagging can be performed by a single pass under a unify classification framework (Ng and Low, 2004). In the rest of the paper, we call this operation mode Joint S&T. Experiments of Ng and Low (2004) shown that, compared with performing segmentation and POS tagging one at a time, Joint S&T can achieve higher accuracy not only on segmentation but also on POS tagging.

Besides the usual local features such as the character-based ones (Xue and Shen, 2003; Ng and Low, 2004), many non-local features related to POSs or words can also be employed to improve performance. However, as such features are generated dynamically during the decoding procedure, incorporating these features directly into the classifier results in problems. First, the classifier's feature space will grow much rapidly, which is apt to overfit on training corpus. Second, the variance of non-local features caused by the model evolution during the training procedure will hurt the parameter tuning. Last but not the lest, since the current predication relies on the results of prior predications, exact inference by dynamic programming can't be obtained, and then we have to maintain a n-best candidate list at each considering position, which also evokes the potential risk of depressing the parameter tuning procedure. As a result, many theoretically useful features such as higher-order word- or POS- grams can not be utilized efficiently.

A widely used approach of using non-local features is the well-known reranking technique, which has been proved effective in many NLP tasks, for instance, syntactic parsing and machine

Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008), pages 385–392
Manchester, August 2008

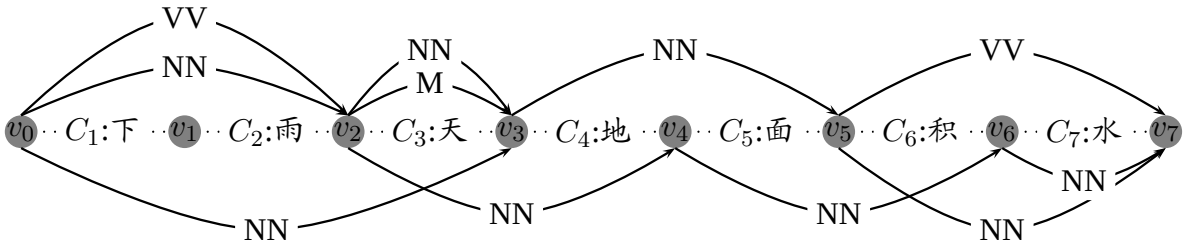

Figure 1: Pruned word lattice as directed graph. The character sequence we choose is "下-雨-天-地-面-积-水". For clarity, we represent each subsequence-POS pair as a single edge, while ignore the corresponding scores of the edges.

translation (Collins, 2000; Huang, 2008), etc. Especially, Huang (2008) reranked the packed forest, which contains exponentially many parses. Inspired by his work, we propose word lattice reranking, a strategy that reranks the pruned word lattice outputted by a baseline classifier, rather than only a n-best list. Word lattice, a directed graph as shown in Figure 1, is a packed structure that can represent many possibilities of segmentation and POS tagging. Our experiments on the Penn Chinese Treebank 5.0 show that, reranking on word lattice gains obvious improvement over the baseline classifier and the reranking on n-best list. Compared against the baseline, we obtain an error reduction of 11.9% on segmentation, and 16.3% on Joint S&T.

2 Word Lattice

Formally, a word lattice L is a directed graph $\langle V, E\rangle$, where V is the node set, and E is the edge set. Suppose the word lattice is for sentence $C_{1:n} = C_1..C_n$, node $v_i \in V$ ($i = 1..n-1$) denotes the position between C_i and C_{i+1}, while v_0 before C_1 is the source node, and v_n after C_n is the sink node. An edge $e \in E$ departs from v_b and arrives at v_e ($0 \le b < e \le n$), it covers a subsequence of $C_{1:n}$, which is recognized as a possible word. Considering Joint S&T, we label each edge a POS tag to represent a word-POS pair. A series of adjoining edges forms a path, and a path connecting the source node and the sink node is called diameter, which indicates a specific pattern of segmentation and POS tagging. For a diameter d, $|d|$ denotes the length of d, which is the count of edges contained in this diameter. In Figure 1, the path $p' = \widehat{v_0\ v_3} \rightarrow \widehat{v_3\ v_5} \rightarrow \widehat{v_5\ v_7}$ is a diameter, and $|p'|$ is 3.

2.1 Oracle Diameter in Lattice

Given a sentence s, its reference r and pruned word lattice L generated by the baseline classifier, the oracle diameter d^* of L is define as the diameter most similar to r. With F-measure as the scoring function, we can identify d^* using the algorithm depicted in Algorithm 1, which is adapted to lexical analysis from the forest oracle computation of Huang (2008).

Before describe this algorithm in detail, we depict the key point for finding the oracle diameter. Given the system's output y and the reference y^*, using $|y|$ and $|y^*|$ to denote word counts of them respectively, and $|y \cap y^*|$ to denote matched word count of $|y|$ and $|y^*|$, F-measure can be computed by:

$$F(y, y^*) = \frac{2PR}{P+R} = \frac{2|y \cap y^*|}{|y| + |y^*|} \qquad (1)$$

Here, $P = \frac{|y \cap y^*|}{|y|}$ is precision, and $R = \frac{|y \cap y^*|}{|y^*|}$ is recall. Notice that $F(y, y^*)$ isn't a linear function, we need access the largest $|y \cap y^*|$ for each possible $|y|$ in order to determine the diameter with maximum F, or another word, we should know the maximum matched word count for each possible diameter length.

The algorithm shown in Algorithm 1 works in a dynamic programming manner. A table node $T[i, j]$ is defined for sequence span $[i, j]$, and it has a structure S to remember the best $|y_{i:j} \cap y^*_{i:j}|$ for each $|y_{i:j}|$, as well as the back pointer for this best choice. The for-loop in line $2-14$ processes for each node $T[i, j]$ in a shorter-span-first order. Line $3-7$ initialize $T[i, j]$ according to the reference r and the word lattice's edge set $L \cdot E$. If there exists an edge e in $L \cdot E$ covering the span $[i, j]$, then we

Algorithm 1 Oracle Diameter, à la Huang (2008, Sec. 4.1).

1: **Input**: sentence s, reference r and lattice L
2: **for** $[i, j] \subseteq [1, |s|]$ in topological order **do**
3: **if** $\exists e \in L \cdot E$ s.t. e spans from i to j **then**
4: **if** $e \cdot label$ exists in r **then**
5: $T[i, j] \cdot S[1] \leftarrow 1$
6: **else**
7: $T[i, j] \cdot S[1] \leftarrow 0$
8: **for** k s.t. $T[i, k-1]$ and $T[k, j]$ defined **do**
9: **for** p s.t. $T[i, k-1] \cdot S[p]$ defined **do**
10: **for** q s.t. $T[k, j] \cdot S[q]$ defined **do**
11: $n \leftarrow T[i, k-1] \cdot S[p] + T[k, j] \cdot S[q]$
12: **if** $n > T[i, j] \cdot S[p+q]$ **then**
13: $T[i, j] \cdot S[p+q] \leftarrow n$
14: $T[i, j] \cdot S[p+q] \cdot bp \leftarrow \langle k, p, q \rangle$
15: $t \leftarrow argmax_t \frac{2 \times T[1,|s|] \cdot S[t]}{t+|r|}$
16: $d^* \leftarrow Tr(T[1, |s|] \cdot S[t].bp)$
17: **Output:** oracle diameter: d^*

define $T[i, j]$, otherwise we leave this node undefined. In the first situation, we initialize this node's S structure according to whether the word-POS pair of e is in the reference (line $4-7$). Line $8-14$ update $T[i, j]$'s S structure using the S structures from all possible child-node pair, $T[i, k-1]$ and $T[k, j]$. Especially, line $9-10$ enumerate all combinations of p and q, where p and q each represent a kind of diameter length in $T[i, k-1]$ and $T[k, j]$. Line $12-14$ refreshes the structure S of node $T[i, j]$ when necessary, and meanwhile, a back pointer $\langle k, p, q \rangle$ is also recorded. When the dynamic programming procedure ends, we select the diameter length t of the top node $T[1, |s|]$, which maximizes the F-measure formula in line 15, then we use function Tr to find the oracle diameter d^* by tracing the back pointer bp.

2.2 Generation of the Word Lattice

We can generate the pruned word lattice using the baseline classifier, with a slight modification. The classifier conducts decoding by considering each character in a left-to-right fashion. At each considering position i, the classifier enumerates all candidate results for subsequence $C_{1:i}$, by attaching each current candidate word-POS pair p to the tail of each candidate result at p's prior position, as the endmost of the new generated candidate. We give each p a score, which is the highest, among all $C_{1:i}$'s candidates that have p as their endmost. Then we select N word-POS pairs with the highest scores, and insert them to the lattice's edge set. This approach of selecting edges implies that, for the lattice's node set, we generate a node v_i at each position i. Because N is the limitation on the count

Algorithm 2 Lattice generation algorithm.

1: **Input**: character sequence $C_{1:n}$
2: $E \leftarrow \emptyset$
3: **for** $i \leftarrow 1$ .. n **do**
4: $cands \leftarrow \emptyset$
5: **for** $l \leftarrow 1$ .. min(i, K) **do**
6: $w \leftarrow C_{i-l+1:i}$
7: **for** $t \in POS$ **do**
8: $p \leftarrow \langle w, t \rangle$
9: $p \cdot score \leftarrow Eval(p)$
10: $s \leftarrow p \cdot score + Best[i-l]$
11: $Best[i] \leftarrow max(s, Best[i])$
12: insert $\langle s, p \rangle$ into $cands$
13: sort $cands$ according to s
14: $E \leftarrow E \cup cands[1..N] \cdot p$
15: **Output:** edge set of lattice: E

of edges that point to the node at position i, we call this pruning strategy *in-degree pruning*. The generation algorithm is shown in Algorithm 2.

Line $3-14$ consider each character C_i in sequence, $cands$ is used to keep the edges closing at position i. Line 5 enumerates the candidate words ending with C_i and no longer than K, where K is 20 in our experiments. Line 5 enumerates all POS tags for the current candidate word w, where POS denotes the POS tag set. Function $Eval$ in line 9 returns the score for word-POS pair p from the baseline classifier. The array $Best$ preserve the score for sequence $C_{1:i}$'s best labelling results. After all possible word-POS pairs (or edges) considered, line $13-14$ select the N edges we want, and add them to edge set E.

Though this pruning strategy seems relative rough — simple pruning for edge set while no pruning for node set, we still achieve a promising improvement by reranking on such lattices. We believe more elaborate pruning strategy will results in more valuable pruned lattice.

3 Reranking

A unified framework can be applied to describing reranking for both n-best list and pruned word lattices (Collins, 2000; Huang, 2008). Given the candidate set $cand(s)$ for sentence s, the reranker selects the best item $\hat{y}$ from $cand(s)$:

$$\hat{y} = \underset{y \in cand(s)}{\operatorname{argmax}} \mathbf{w} \cdot \mathbf{f}(y) \quad (2)$$

For reranking n-best list, $cand(s)$ is simply the set of n best results from the baseline classifier. While for reranking word lattice, $cand(s)$ is the set of all diameters that are impliedly built in the lattice. $\mathbf{w} \cdot \mathbf{f}(y)$ is the dot product between a feature vector $\mathbf{f}$ and a weight vector $\mathbf{w}$, its value is used to

Algorithm 3 Perceptron training for reranking

Input: Training examples$\{cand(s_i), y_i^*\}_{i=1}^N$
$\mathbf{w} \leftarrow \mathbf{0}$
for $t \leftarrow 1$ .. T **do**
  for $i \leftarrow 1$ .. N **do**
    $\hat{y} \leftarrow \arg max_{y \in cand(s_i)} \mathbf{w} \cdot \mathbf{f}(y)$
    if $\hat{y} \neq y_i^*$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(y_i^*) - \mathbf{f}(\hat{y})$
Output: Parameters: $\mathbf{w}$

Non-local Template	Comment
W_0T_0	current word-POS pair
W_{-1}	word 1-gram before W_0T_0
T_{-1}	POS 1-gram before W_0T_0
$T_{-2}T_{-1}$	POS 2-gram before W_0T_0
$T_{-3}T_{-2}T_{-1}$	POS 3-gram before W_0T_0

Table 1: Non-local feature templates used for reranking

rerank $cand(s)$. Following usual practice in parsing, the first feature $f_1(y)$ is specified as the score outputted by the baseline classifier, and its value is a real number. The other features are non-local ones such as word- and POS- n-grams extracted from candidates in n-best list (for n-best reranking) or diameters (for word lattice reranking), and they are $0-1$ valued.

3.1 Training of the Reranker

We adopt the perceptron algorithm (Collins, 2002) to train the reranker. as shown in Algorithm 3. We use a simple refinement strategy of "averaged parameters" of Collins (2002) to alleviate overfitting on the training corpus and obtain more stable performance.

For every training example $\{cand(s_i), y_i^*\}$, y_i^* denotes the best candidate in $cand(s_i)$. For n-best reranking, the best candidate is easy to find, whereas for word lattice reranking, we should use the algorithm in Algorithm 1 to determine the oracle diameter, which represents the best candidate result.

3.2 Non-local Feature Templates

The non-local feature templates we use to train the reranker are listed in Table 1. Notice that all features generated from these templates don't contain "future" words or POS tags, it means that we only use current or history word- or POS- n-grams to evaluate the current considering word-POS pair. Although it is possible to use "future" information in n-best list reranking, it's not the same when we rerank the pruned word lattice. As we have to traverse the lattice topologically, we face difficulty in utilizing the information ahead of the current considering node.

Algorithm 4 Cube pruning for non-local features.

1: **function** CUBE(L)
2: **for** $v \in L \cdot V$ in topological order **do**
3: NBEST(v)
4: **return** $\mathbf{D_{v_{sink}}}[1]$
5: **procedure** NBEST(v)
6: $heap \leftarrow \emptyset$
7: **for** v' topologically before v **do**
8: $\mathbf{\Delta} \leftarrow$ all edges from v' to v
9: $p \leftarrow \langle \mathbf{D_{v'}}, \mathbf{\Delta} \rangle$
10: $\langle p, \mathbf{1} \rangle \cdot score \leftarrow Eval(p, \mathbf{1})$
11: PUSH($\langle p, \mathbf{1} \rangle, heap$)
12: HEAPIFY($heap$)
13: $buf \leftarrow \emptyset$
14: **while** $|heap| > 0$ and $|buf| < N$ **do**
15: $item \leftarrow$ POP-MAX($heap$)
16: append $item$ to buf
17: PUSHSUCC($item, heap$)
18: sort buf to $\mathbf{D_v}$
19: **procedure** PUSHSUCC($\langle p, \mathbf{j} \rangle, heap$)
20: p is $\langle \mathbf{vec_1}, \mathbf{vec_2} \rangle$
21: **for** $i \leftarrow 1..2$ **do**
22: $\mathbf{j'} \leftarrow \mathbf{j} + \mathbf{b^i}$
23: **if** $|\mathbf{vec_i}| \geq j_i'$ **then**
24: $\langle p, \mathbf{j'} \rangle \cdot score \leftarrow Eval(p, \mathbf{j'})$
25: PUSH($\langle p, \mathbf{j'} \rangle, heap$)

3.3 Reranking by Cube Pruning

Because of the non-local features such as word- and POS- n-grams, the reranking procedure is similar to machine translation decoding with intergrated language models, and should maintain a list of N best candidates at each node of the lattice. To speed up the procedure of obtaining the N best candidates, following Huang (2008, Sec. 3.3), we adapt the cube pruning method from machine translation (Chiang, 2007; Huang and Chiang 2007) which is based on efficient k-best parsing algorithms (Huang and Chiang, 2005).

As shown in Algorithm 4, cube pruning works topologically in the pruned word lattice, and maintains a list of N best derivations at each node. When deducing a new derivation by attaching a current word-POS pair to the tail of a antecedent derivation, a function $Eval$ is used to compute the new derivation's score (line 10 and 24). We use a max-heap $heap$ to hold the candidates for the next-best derivation. Line $7-11$ initialize $heap$ to the set of top derivations along each deducing source, the vector pair $\langle \mathbf{D_{v_{head}}}, \mathbf{\Delta} \rangle$.Here, $\mathbf{\Delta}$ denotes the vector of current word-POS pairs, while $\mathbf{D_{v_{head}}}$ denotes the vector of N best derivations at $\mathbf{\Delta}$'s antecedent node. Then at each iteration,

Non-lexical-target	Instances
C_n $(n = -2..2)$	C_{-2}=下, C_{-1}=雨, C_0=天, C_1=地, C_2=面
C_nC_{n+1} $(n = -2..1)$	$C_{-2}C_{-1}$=下雨, $C_{-1}C_0$=雨天, C_0C_1=天地, C_1C_2=地面
$C_{-1}C_1$	$C_{-1}C_1$=雨地
Lexical-target	**Instances**
C_0C_n $(n = -2..2)$	C_0C_{-2}=天下, C_0C_{-1}=天雨, C_0C_0=天天, C_0C_1=天地, C_0C_2=天面
$C_0C_nC_{n+1}$ $(n = -2..1)$	$C_0C_{-2}C_{-1}$=天下雨, $C_0C_{-1}C_0$=天雨天, $C_0C_0C_1$=天天地, $C_0C_1C_2$=天地面
$C_0C_{-1}C_1$	$C_0C_{-1}C_1$ = 天雨地

Table 2: Feature templates and instances. Suppose we consider the third character "天" in the sequence "下雨天地面".

we pop the best derivation from $heap$ (line 15), and push its successors into $heap$ (line 17), until we get N derivations or $heap$ is empty. In line 22 of function PUSHSUCC, $\mathbf{j}$ is a vector composed of two index numbers, indicating the two candidates' indexes in the two vectors of the deducing source p, where the two candidates are selected to deduce a new derivation. $\mathbf{j}'$ is a increment vector, whose ith dimension is 1, while others are 0. As non-local features (word- and POS- n-grams) are used by function $Eval$ to compute derivation's score, the derivations extracted from $heap$ may be out of order. So we use a buffer buf to keep extracted derivations (line 16), then sort buf and put its first N items to $\mathbf{D_v}$ (line 18).

4 Baseline Perceptron Classifier

4.1 Joint S&T as Classification

Following Jiang et al. (2008), we describe segmentation and Joint S&T as below:

For a given Chinese sentence appearing as a character sequence:

$$C_{1:n} = C_1\ C_2\ ..\ C_n$$

the goal of segmentation is splitting the sequence into several subsequences:

$$C_{1:e_1}\ C_{e_1+1:e_2}\ ..\ C_{e_{m-1}+1:e_m}$$

While in Joint S&T, each of these subsequences is labelled a POS tag:

$$C_{1:e_1}/t_1\ C_{e_1+1:e_2}/t_2\ ..\ C_{e_{m-1}+1:e_m}/t_m$$

Where $C_i(i = 1..n)$ denotes a character, $C_{l:r}(l \leq r)$ denotes the subsequence ranging from C_l to C_r, and $t_i(i = 1..m, m \leq n)$ denotes the POS tag of $C_{e_{i-1}+1:e_i}$.

If we label each character a positional tag indicating its relative position in an expected subsequence, we can obtain the segmentation result accordingly. As described in Ng and Low (2004) and Jiang et al. (2008), we use s indicating a *single*-character word, while b, m and e indicating the *begin*, *middle* and *end* of a word respectively. With these positional tags, the segmentation transforms to a classification problem. For Joint S&T, we expand positional tags by attaching POS to their tails as postfix. As each tag now contains both positional- and POS- information, Joint S&T can also be resolved in a classification style framework. It means that, a subsequence is a word with POS t, only if the positional part of the tag sequence conforms to s or bm^*e pattern, and each element in the POS part equals to t. For example, a tag sequence $b\_NN\ m\_NN\ e\_NN$ represents a three-character word with POS tag NN.

4.2 Feature Templates

The features we use to build the classifier are generated from the templates of Ng and Low (2004). For convenience of comparing with other, they didn't adopt the ones containing external knowledge, such as punctuation information. All their templates are shown in Table 2. C denotes a character, while its subscript indicates its position relative to the current considering character(it has the subscript 0).

The table's upper column lists the templates that immediately from Ng and Low (2004). they named these templates *non-lexical-target* because predications derived from them can predicate without considering the current character C_0. Templates called *lexical-target* in the column below are introduced by Jiang et al. (2008). They are generated by adding an additional field C_0 to each *non-lexical-target* template, so they can carry out predication not only according to the context, but also according to the current character itself.

Notice that features derived from the templates in Table 2 are all local features, which means all features are determined only by the training instances, and they can be generated before the training procedure.

Algorithm 5 Perceptron training algorithm.

Input: Training examples (x_i, y_i)
$\alpha \leftarrow \mathbf{0}$
for $t \leftarrow 1$ .. T **do**
  for $i \leftarrow 1$ .. N **do**
    $z_i \leftarrow \arg max_{z \in \mathbf{GEN}(x_i)} \mathbf{\Phi}(x_i, z) \cdot \alpha$
    if $z_i \neq y_i$ **then**
      $\alpha \leftarrow \alpha + \mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, z_i)$
Output: Parameters: α

4.3 Training of the Classifier

Collins (2002)'s perceptron training algorithm were adopted again, to learn a discriminative classifier, mapping from inputs $x \in X$ to outputs $y \in Y$. Here x is a character sequence, and y is the sequence of classification result of each character in x. For segmentation, the classification result is a positional tag, while for Joint S&T, it is an extended tag with POS information. X denotes the set of character sequence, while Y denotes the corresponding set of tag sequence.

According to Collins (2002), the function $\mathbf{GEN}(x)$ generates all candidate tag sequences for the character sequence x , the representation $\mathbf{\Phi}$ maps each training example $(x, y) \in X \times Y$ to a feature vector $\mathbf{\Phi}(x, y) \in R^d$, and the parameter vector $\alpha \in R^d$ is the weight vector corresponding to the expected perceptron model's feature space. For a given input character sequence x, the mission of the classifier is to find the tag sequence $F(x)$ satisfying:

$$F(x) = \underset{y \in \mathbf{GEN}(x)}{\text{argmax}} \ \mathbf{\Phi}(x, y) \cdot \alpha \qquad (3)$$

The inner product $\mathbf{\Phi}(x, y) \cdot \alpha$ is the score of the result y given x, it represents how much plausibly we can label character sequence x as tag sequence y. The training algorithm is depicted in Algorithm 5. We also use the "averaged parameters" strategy to alleviate overfitting.

5 Experiments

Our experiments are conducted on the Penn Chinese Treebank 5.0 (CTB 5.0). Following usual practice of Chinese parsing, we choose chapters $1-260$ (18074 sentences) as the training set, chapters $301-325$ (350 sentences) as the development set, and chapters $271-300$ (348 sentences) as the final test set. We report the performance of the baseline classifier, and then compare the performance of the word lattice reranking against the

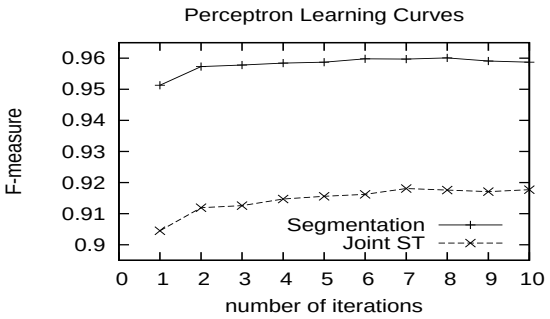

Figure 2: Baseline averaged perceptron learning curves for segmentation and Joint S&T.

n-best reranking, based on this baseline classifier. For each experiment, we give accuracies on segmentation and Joint S&T. Analogous to the situation in parsing, the accuracy of Joint S&T means that, a word-POS is recognized only if both the positional- and POS- tags are correctly labelled for each character in the word's span.

5.1 Baseline Perceptron Classifier

The perceptron classifier are trained on the training set using features generated from the templates in Table 2, and the development set is used to determine the best parameter vector. Figure 2 shows the learning curves for segmentation and Joint S&T on the development set. We choose the averaged parameter vector after 7 iterations for the final test, this parameter vector achieves an F-measure of 0.973 on segmentation, and 0.925 on Joint S&T. Although the accuracy on segmentation is quite high, it is obviously lower on Joint S&T. Experiments of Ng and Low (2004) on CTB 3.0 also shown the similar trend, where they obtained F-measure 0.952 on segmentation, and 0.919 on Joint S&T.

5.2 Preparation for Reranking

For n-best reranking, we can easily generate n best results for every training instance, by a modification for the baseline classifier to hold n best candidates at each considering point. For word lattice reranking, we use the algorithm in Algorithm 2 to generate the pruned word lattice. Given a training instance s_i, its n best result list or pruned word lattice is used as a reranking instance $cand(s_i)$, the best candidate result (of the n best list) or oracle diameter (of the pruned word lattice) is the reranking target y_i^*. We find the best result of the n best results simply by computing each result's

F-measure, and we determine the oracle diameter of the pruned word lattice using the algorithm depicted in Algorithm 1. All pairs of $cand(s_i)$ and y_i^* deduced from the baseline model's training instances comprise the training set for reranking. The development set and test set for reranking are obtained in the same way. For the reranking training set $\{cand(s_i), y_i^*\}_{i=1}^N$, $\{y_i^*\}_{i=1}^N$ is called oracle set, and the F-measure of $\{y_i^*\}_{i=1}^N$ against the reference set is called oracle F-measure. We use the oracle F-measure indicating the utmost improvement that an reranking algorithm can achieve.

5.3 Results and Analysis

The flows of the n-best list reranking and the pruned word lattice reranking are similar to the training procedure for the baseline classifier. The training set for reranking is used to tune the parameter vector of the reranker, while the development set for reranking is used to determine the optimal number of iterations for the reranker's training procedure.

We compare the performance of the word lattice reranking against the n-best list reranking. Table 3 shows the experimental results. The upper four rows are the experimental results for n-best list reranking, while the four rows below are for word lattice reranking. In n-best list reranking, with list size 20, the oracle F-measure on Joint S&T is 0.9455, and the reranked F-measure is 0.9280. When list size grows up to 50, the oracle F-measure on Joint S&T jumps to 0.9552, while the reranked F-measure becomes 0.9302. However, when n grows to 100, it brings tiny improvement over the situation of $n = 50$. In word lattice reranking, there is a trend similar to that in n-best reranking, the performance difference between $in\_degree = 2$ and $in\_degree = 5$ is obvious, whereas the setting $in\_degree = 10$ does not obtain a notable improvement over the performance of $in\_degree = 5$. We also notice that even with a relative small in_degree limitation, such as $in\_degree = 5$, the oracle F-measures for segmentation and Joint S&T both reach a quite high level. This indicates the pruned word lattice contains much more possibilities of segmentation and tagging, compared to n-best list.

With the setting $in\_degree = 5$, the oracle F-measure on Joint S&T reaches 0.9774, and the reranked F-measure climbs to 0.9336. It achieves an error reduction of 16.3% on Joint S&T, and an error reduction of 11.9% on segmentation, over the baseline classifier. While for n-best reranking with setting $n = 50$, the Joint S&T's error reduction is 6.9% , and the segmentation's error reduction is 8.9%. We can see that reranking on pruned word lattice is a practical method for segmentation and POS tagging. Even with a much small data representation, it obtains obvious advantage over the n-best list reranking.

n-best	Ora Seg	Tst Seg	Ora S&T	Tst S&T
20	0.9827	0.9749	0.9455	0.9280
50	0.9903	0.9754	0.9552	0.9302
100	0.9907	0.9755	0.9558	0.9305
Degree	**Ora Seg**	**Rnk Seg**	**Ora S&T**	**Rnk S&T**
2	0.9898	0.9753	0.9549	0.9296
5	0.9927	0.9774	0.9768	0.9336
10	0.9934	0.9774	0.9779	0.9337

Table 3: Performance of n-best list reranking and word lattice reranking. n-best: the size of the n-best list for n-best list reranking; Degree: the in degree limitation for word lattice reranking; Ora Seg: oracle F-measure on segmentation of n-best lists or word lattices; Ora S&T: oracle F-measure on Joint S&T of n-best lists or word lattices; Rnk Seg: F-measure on segmentation of reranked result; Rnk S&T: F-measure on Joint S&T of reranked result

Comparing between the baseline and the two reranking techniques, We find the non-local information such as word- or POS- grams do improve accuracy of segmentation and POS tagging, and we also find the reranking technique is effective to utilize these kinds of information. As even a small scale n-best list or pruned word lattice can achieve a rather high oracle F-measure, reranking technique, especially the word lattice reranking would be a promising refining strategy for segmentation and POS tagging. This is based on this viewpoint: On the one hand, compared with the initial input character sequence, the pruned word lattice has a quite smaller search space while with a high oracle F-measure, which enables us to conduct more precise reranking over this search space to find the best result. On the other hand, as the structure of the search space is approximately outlined by the topological directed architecture of pruned word lattice, we have a much wider choice for feature selection, which means that we would be able to utilize not only features topologically before the current considering position, just like those depicted in Table 2 in section 4, but also information topologically after it, for example the next word W_1 or the next POS tag T_1. We believe the pruned word

lattice reranking technique will obtain higher improvement, if we develop more precise reranking algorithm and more appropriate features.

6 Conclusion

This paper describes a reranking strategy called word lattice reranking. As a derivation of the forest reranking of Huang (2008), it performs reranking on pruned word lattice, instead of on n-best list. Using word- and POS- gram information, this reranking technique achieves an error reduction of 16.3% on Joint S&T, and 11.9% on segmentation, over the baseline classifier, and it also outperforms reranking on n-best list. It confirms that word lattice reranking can effectively use non-local information to select the best candidate result, from a relative small representation structure while with a quite high oracle F-measure. However, our reranking implementation is relative coarse, and it must have many chances for improvement. In future work, we will develop more precise pruning algorithm for word lattice generation, to further cut down the search space while maintaining the oracle F-measure. We will also investigate the feature selection strategy under the word lattice architecture, for effective use of non-local information.

Acknowledgement

This work was supported by National Natural Science Foundation of China, Contracts 60736014 and 60573188, and 863 State Key Project No. 2006AA010108. We show our special thanks to Liang Huang for his valuable suggestions.

References

Collins, Michael. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning*, pages 175–182.

Collins, Michael. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*, pages 1–8, Philadelphia, USA.

Fine, Shai, Yoram Singer, and Naftali Tishby. 1998. The hierarchical hidden markov model: Analysis and applications. In *Machine Learning*, pages 32–41.

Huang, Liang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*.

Jiang, Wenbin, Liang Huang, Yajuan Lv, and Qun Liu. 2008. A cascaded linear model for joint chinese word segmentation and part-of-speech tagging. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*.

Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 282–289, Massachusetts, USA.

Ng, Hwee Tou and Jin Kiat Low. 2004. Chinese part-of-speech tagging: One-at-a-time or all-at-once? word-based or character-based? In *Proceedings of the Empirical Methods in Natural Language Processing Conference*.

Rabiner, Lawrence. R. 1989. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of IEEE*, pages 257–286.

Ratnaparkhi and Adwait. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*.

Xue, Nianwen and Libin Shen. 2003. Chinese word segmentation as lmr tagging. In *Proceedings of SIGHAN Workshop*.