Generalization of Words for Chinese Dependency Parsing

Xianchao Wu, Jie Zhou, Yu Sun, Zhanyi Liu, Dianhai Yu, Hua Wu, Haifeng Wang
Baidu Inc.
{wuxianchao,zhoujie01,sunyu02,liuzhanyi,yudianhai,wu_hua,wanghaifeng}@baidu.com

Abstract

In this paper, we investigate the influence of generalization of words to the accuracies of Chinese dependency parsing. Specially, in our shift-reduce parser, we use a neural language model based word embedding (NLMWE) method (Bengio et al., 2003) to generate *distributed word feature vectors* and then perform K-means based word clustering to generate word classes. We designed feature templates by making use of words, part-of-speech (POS) tags, coarse-grained POS (CPOS) tags, NLMWE-based word classes and their combinations. NLMWE-based word classes is shown to be an important supplement of POS-tags, especially when POS-tags are automatically generated. Experiments on a Query treebank, CTB5 and CTB7 show that the combinations of features from CPOS-tags, POS-tags, and NLMWE-based word classes yield the best unlabelled attachment scores (UASs). Our final UAS$-p$ (excluding punctuations) of 86.79% on the CTB5 test set is comparable to state-of-the-art results. Our final UAS$-p$ of 86.80% and 87.05% on the CTB7 Stanford dependency test set and original test set is significantly better than three well known open-source dependency parsers.

1 Introduction

Current dependency parsing framework is facing the following challenge, training the model using manually annotated treebanks and then apply the model to the whole Web. In terms of words, it is not possible for the treebank to cover all the words in the Web. Given a test sentence, how can we expect the parser to output a correct tree if there are out-of-vocabulary (OOV) words (compared to the training data of the treebank) and/or the POS-tags are wrongly annotated?

Words need to be *generalized* to solve this problem in a sense. Indeed, POS-tag itself is a way to generalize words into word classes. This is because POS-taggers can be trained on larger-scale data compared with treebanks. Annotating trees is far more difficult than annotating POS-tags. Considering that unsupervised word clustering methods can make use of TB/PB-level Web data, these approaches have been shown to be helpful for dependency parsing (Koo et al., 2008).

In this paper, we investigate the influence of generalization of words to the accuracies of Chinese dependency parsing. Specially, in our shift-reduce parser, we use a neural language model based word embedding method (Bengio et al., 2003) to generate *distributed word feature vectors* and then perform K-means based word clustering (Yu et al., 2013) to generate word classes. Our usage of word embedding is in line with Turian et al. (2010) and Yu et al. (2013), who study the effects of different clustering algorithms for POS tagging and named entity recognition (NER). We designed feature templates by making use of words, POS tags, CPOS tags, NLMWE-based word classes and their combinations. NLMWE-based word classes is shown to be an important supplement of POS-tags. Experiments on a Query treebank, CTB5 and CTB7 show that the combinations of features from CPOS-tags, POS-tags, and NLMWE-based word classes yield the best UASs.

1.1 Shift-reduce parsing

We use a transition-based shift-reduce parser (Kudo and Matsumoto, 2002; Nivre, 2003; Nivre et al., 2006; Huang and Sagae, 2010) to

perform all the experiments in this paper. In a typical transition-based parsing process, the input words are stored in a queue and partially built dependency structures (e.g., sub-trees) are organized by a configuration (or state). A parser configuration (or, state) can be represented by a tuple $< S, N, A >$, where S is the stack, N is the queue of incoming words, and A is the set of dependency arcs that have been built. A set of shift-reduce actions are defined, which are used to construct new dependency arcs by connecting the top word of the queue and the top word of the stack. We adopt the arc-standard system (Nivre, 2008), whose actions include:

- *shift*, which removes the top word in the queue and pushes it onto the top of the stack;
- *left-arc*, which pops the top item off the stack, and adds it as a modifier to the front of the queue;
- *right-arc*, which removes the front of the queue, and adds it as a modifier to the top of the stack. In addition, the top of the stack is popped and added to the front of the queue.

We follow Kudo and Matsumoto (2002) and use the Support Vector Machines (SVMs) for action classification training and beam search (Zhang and Clark, 2008) for decoding.

2 Neural Language Model Based Word Embedding

Following (Bengio et al., 2003), we use a neural network with two hidden layers to learn *distributed word feature vectors* from large-scale training data. Recall that, the goal of statistical language modelling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the *curse of dimensionality*: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training (so called OOV words and/or sequences). N-gram based approach obtains generalization by concatenating very short overlapping sequences seen in the training set. Bengio et al. (2003) propose to fight the curse of dimensionality by learning a *distributed* representation (of feature vectors) for words which allows each training sentence to inform the model about an exponential number of semantically neighbouring sentences. The model learns simultaneously (1) a distributed representation for each word along with (2) the probability function for word sequences, expressed in terms of these representations.

The general process of neural language model based word embedding is as follows:

- associate with each word in the vocabulary a distributed word feature vector (a real valued vector in R^m);
- express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence; and,
- learn simultaneously the word feature vectors and the parameters of that probability function.

The training set of the NLMWE model is a sequence $w_1, ..., w_T$ of words $w_t \in V$, where the vocabulary V is a large yet finite set. The objective is to learn a model $f(w_t, ..., w_{t-n+1}) = P(w_t|w_1^{t-1})$, so that that the model gives high out-of-sample likelihood. The only constraint on the model is that for any choice of w_1^{t-1}, $\sum_{i=1}^{|V|} f(i, w_{t-1}, ..., w_{t-n+1}) = 1$ with $f > 0$. By the product of these conditional probabilities, one obtains a model of the joint probability of sequences of words.

Function $f(w_t, ..., w_{t-n+1}) = P(w_t|w_1^{t-1})$ is decomposed into two parts:

- A mapping C from any element i of V to a real vector $C(i) \in R^m$. It represents the *distributed feature vectors* associated with each word in the vocabulary. In practice, C is represented by a $|V| \times m$ matrix of free parameters.
- The probability function over words, expressed with C: a function g maps an input sequence of feature vectors for words in context, $(C(w_{t-n+1}), ..., C(w_{t-1}))$, to a conditional probability distribution over words in V for the next word w_t . The output of g is a vector whose i-th element estimates the probability $P(w_t = i|w_1^{t-1})$ as in Figure 1 in (Bengio et al., 2003).

Following (Bengio et al., 2003), in the real implementation, we speed up with both parallel data and parallel parameter estimation. We finally use the well-known K-means clustering algorithm based on the

distributed word feature vectors for word clustering. We separately set the number of word classes to be 100, 500, and 1,000 in our experiments.

3 Feature Templates

At each step during shift-reducing, a parser configuration (or, state) can be represented by a tuple $< S, N, A >$. We denote the top of stack with S_0, the front items from the queue with N_0, N_1, N_2, and N_3, the leftmost and rightmost modifiers of S_0 (if any) with S_{0l} and S_{0r}, respectively, and the leftmost modifier of N_0 (if any) with N_{0l} (refer to Figure 1). The baseline feature templates without any word class level information (such as POS-tags) are shown in Table 1. These features are mostly taken from Zhang and Clark (2008), Huang and Sagae (2010), and Zhang and Nivre (2011). In this table, w, l and d represents the word, dependency label, and the distance between S_0 and N_0, respectively. For example, S_0wN_0w represents the feature template that takes the word of S_0, and combines it with the word of N_0.

In Table 1, $(S/N)_{0l2}$, $(S/N)_{0r2}$, and $(S/N)_{0rn}$ refer to the second leftmost modifier, the second rightmost modifier, and the right nearest modifier of $(S/N)_0$, respectively. It should be mentioned that, for the arc-standard algorithm used in this paper, S_0 or N_0 never contain a head word. The reason is that, once the head is found for a node, that node will be hidden under the head and being removed from the stack or the queue[1].

Table 2 lists the features templates that are related to POS-tags and their combination with words and dependency labels. In the table, $(S/N)_{0ll}$ and $(S/N)_{0rr}$ stand for the left(ll)/right(rr)-hand-side neighbour word of S_0/N_0 in the input sentence for training or decoding. For example, features such as POS-tags of *bi-siblings* of N_0 (e.g., $N_{0l2}pN_{0r2}p$) and S_0 (e.g., $S_{0r2}pS_{0r}p$) are included in the combination of two feature templates. These *bi-sibling* features together with $(S/N)_0$ (e.g., $N_0N_{0l2}pN_{0r2}p$ and $S_0S_{0r2}pS_{0r}p$) are included in the combination of three feature templates. N_0 (similar with S_0) and its surrounding words, such as neighbour words and child words are shown in Figure 1 for intuitive understanding. For comparison with other levels of word classes, we will directly replace POS-tags (p) by other kind of word classes, such as CPOS-tags and NLMWE-based K-means word clusterings. For example, instead of returning the POS-tag of S_0, S_0p will return CPOS-tag of the word of S_0 in CPOS-tag related feature templates, and return NLMWE-based word class index of the word of S_0 in NLMWE clustering related feature templates.

Besides Table 1 and 2, we further include S_0 and S_1 related features[2] following (Huang and Sagae, 2010). These features are listed in Table 3. As will be shown in Table 16, UASs are improved around 0.2% after appending these feature templates to Table 1 and 2.

single feature templates (14)
$N_{0l}w$, N_0w, N_1w, N_2w, $S_{0l2}l$, $S_{0l2}w$, $S_{0l}l$, $S_{0l}w$, $S_{0r2}l$, $S_{0r2}w$, $S_{0rn}w$, $S_{0r}l$, $S_{0r}w$, S_0w
combination of two feature templates (19)
$N_{0l}wS_0w$, $N_{0rn}wS_0w$, $N_{0r}wS_0w$, $N_0wN_{0l}l$, N_0wN_1w, N_0wN_2w, N_0wN_3w, N_0wd, N_1wN_2w, N_1wN_3w, $N_1wS_{0r}l$, N_2wN_3w, $S_{0l}wN_0w$, $S_{0rn}wN_0w$, $S_{0r}wN_0w$, S_0wN_0w, $S_0wS_{0l}l$, $S_0wS_{0r}l$, S_0wd
combination of three feature templates (8)
$N_0wN_{0l}lN_{0l2}l$, $N_0wN_1wN_2w$, $N_0wN_1wN_3w$, $N_0wN_2wN_3w$, $N_1wN_2wN_3w$, S_0wN_0wd, $S_0wS_{0l}lS_{0l2}l$, $S_0wS_{0r}lS_{0r2}l$
combination of four feature templates (1)
$N_0wN_1wN_2wN_3w$

Table 1: Feature templates related to words (w), dependency labels (l) and the distance between S_0 and N_0 (d).

combination of two feature templates (3)
S_1wS_1p, S_0wS_1w, S_0pS_1p
combination of three feature templates (13)
$S_0wS_0pS_1p$, $S_0pS_1wS_1p$, $S_0wS_1wS_1p$, $S_0wS_0pS_1w$, $S_1pS_0pN_0p$, $S_1pS_0wN_0p$, $S_1pS_{1l}pS_0p$, $S_1pS_{1r}pS_0p$, $S_1pS_0pS_{0r}p$, $S_1pS_{1l}pS_0p$, $S_1pS_{1r}pS_0w$, $S_1pS_0wS_{0l}p$, $S_2pS_1pS_0p$
combination of four feature templates (1)
$S_0wS_0pS_1wS_1p$

Table 3: Feature templates related to S_1 and S_2.

4 Experiments

4.1 Setup

We use three Chinese treebanks in our experiments. The first one is an in-house Chinese Query treebank with 12,028 sentences (averagely 4.04 words per sen-

[1]Thanks one reviewer for pointing this out.

[2]Thanks one reviewer for pointing this out.

single feature templates (13)
$N_{0l2}p$, $N_{0l}p$, N_0p, N_1p, N_2p, N_4p, $S_{0l2}p$, $S_{0l}p$, $S_{0r2}p$, $S_{0r}p$, S_0p, S_1p, S_2p
combination of two feature templates (54)
$N_{0l2}pN_{0r2}p$, $N_{0l2}pN_{0rn}p$, $N_{0l2}pN_{0r}p$, $N_{0l}pN_{0l2}p$, $N_{0l}pN_{0r2}p$, $N_{0l}pN_{0rn}p$, $N_{0l}pN_{0r}p$, $N_{0l}pS_0p$, $N_{0l}pS_0w$, $N_{0l}wS_0p$, $N_{0r2}pN_{0r}p$, $N_{0rn}pN_{0r2}p$, $N_{0rn}pN_{0r}p$, $N_{0rn}pS_0p$, $N_{0rn}pS_0w$, $N_{0rn}wS_0p$, $N_{0r}pS_0p$, $N_{0r}pS_0w$, $N_{0r}wS_0p$, $N_0pN_{0l}l$, N_0pN_1p, N_0pN_2p, N_0pN_3p, N_0pd, N_0wN_0p, N_1pN_2p, N_1pN_3p, N_1pS_0w, N_1wN_1p, N_2pN_3p, $S_{0l2}pS_{0r2}p$, $S_{0l2}pS_{0rn}p$, $S_{0l2}pS_{0r}p$, $S_{0l}pN_0p$, $S_{0l}pN_0w$, $S_{0l}pS_{0l2}p$, $S_{0l}pS_{0r2}p$, $S_{0l}pS_{0rn}p$, $S_{0l}pS_{0r}p$, $S_{0l}wN_0p$, $S_{0r2}pS_{0r}p$, $S_{0rn}pN_0p$, $S_{0rn}pN_0w$, $S_{0rn}pS_{0r2}p$, $S_{0rn}pS_{0r}p$, $S_{0rn}wN_0p$, $S_{0r}pN_0p$, $S_{0r}pN_0w$, $S_{0r}wN_0p$, S_0pN_0p, $S_0pS_{0l}l$, $S_0pS_{0r}l$, S_0pd, S_0wS_0p
combination of three feature templates (43)
$N_{0ll}pN_0pS_0p$, $N_{0ll}pS_0pS_{0rr}p$, $N_0pN_{0l2}pN_{0r2}p$, $N_0pN_{0l2}pN_{0rn}p$, $N_0pN_{0l2}pN_{0r}p$, $N_0pN_{0l}lN_{0l2}l$, $N_0pN_{0l}pN_{0l2}p$, $N_0pN_{0l}pN_{0r2}p$, $N_0pN_{0l}pN_{0rn}p$, $N_0pN_{0l}pN_{0r}p$, $N_0pN_{0r2}pN_{0r}p$, $N_0pN_{0rn}pN_{0r2}p$, $N_0pN_{0rn}pN_{0r}p$, $N_0pN_{0rn}pS_0p$, $N_0pN_{0rr}pS_0p$, $N_0pN_{0r}pS_0p$, $N_0pN_1pN_2p$, $N_0pN_1pN_3p$, $N_0pN_2pN_3p$, $N_0pS_{0ll}pS_0p$, $N_0pS_0pS_{0rr}p$, $N_1pN_2pN_3p$, $S_0pN_0pN_{0l}p$, $S_0pN_0pN_1p$, S_0pN_0pd, $S_0pN_0wN_0p$, $S_0pS_{0l2}pS_{0r2}p$, $S_0pS_{0l2}pS_{0rn}p$, $S_0pS_{0l2}pS_{0r}p$, $S_0pS_{0l}lS_{0l2}l$, $S_0pS_{0l}pN_0p$, $S_0pS_{0l}pS_{0l2}p$, $S_0pS_{0l}pS_{0r2}p$, $S_0pS_{0l}pS_{0rn}p$, $S_0pS_{0l}pS_{0r}p$, $S_0pS_{0rn}pN_0p$, $S_0pS_{0rn}pS_{0r2}p$, $S_0pS_{0rn}pS_{0r}p$, $S_0pS_{0r}lS_{0r2}l$, $S_0pS_{0r}pS_{0r2}p$, $S_0wN_0wN_0p$, $S_0wS_0pN_0p$, $S_0wS_0pN_0w$
combination of four feature templates (6)
$N_{0ll}pN_0pS_{0ll}pS_0p$, $N_{0ll}pN_0pS_0pS_{0rr}p$, $N_0pN_{0rr}pS_{0ll}pS_0p$, $N_0pN_{0rr}pS_0pS_{0rr}p$, $N_0pN_1pN_2pN_3p$, $S_0wS_0pN_0wN_0p$

Table 2: Feature templates related to POS-tags (p) and their combination with words (w) and dependency labels (l).

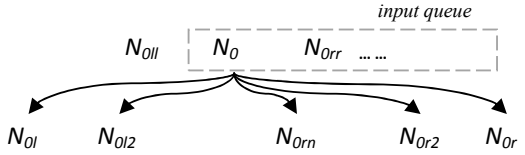

Figure 1: N_0, its neighbour words (N_{0ll} and N_{0rr}), and child words (N_{0l}, N_{0l2}, N_{0rn}, N_{0r2}, and N_{0r}).

tence) for training and 500 sentences (averagely 3.95 words per sentence) for testing.

	Sections	Sentences	Tokens
Train	001-815;1001-1136	16,118	437,859
Dev	886-931;1148-1151	804	20,453
Test	816-885;1137-1147	1,915	50,319

Table 4: Standard split of CTB5 data.

The second one is the Chinese Treebank 5.0[3] (CTB5) (Xue et al., 2007). We follow the standard split of this treebank, as shown in Table 4. The development set is used to tune the hyper-parameter in the SVM classification model for predicting the next sift-reduce actions. We report the unlabelled attachment scores (UASs) of the test set with and without punctuations. We use Penn2Malt toolkit[4] with Chinese head rules[5] to convert the PCFG trees into CoNLL-style[6] dependency trees. Besides the head rules originally used in Penn2Malt, we also independently use the head rules expressed in (Zhang and Clark, 2008) for direct comparison with related works (refer to Table 16). In our experiments, it reveals that using of these two kinds of head rules does not bring a significant differences of UASs.

	Files	Sentences	Tokens
Train	2,083	46,572	1,039,942
Dev	160	2,079	59,955
Test	205	2,796	81,578

Table 5: Statistics of CTB7 data.

The third one is the Chinese Treebank 7.0[7] (CTB7). The statistics is shown in Table 5 by following the standard split of this treebank. Again, the development set is used to tune the hyper-parameter in the SVM classification model. The PCFG tree to CoNLL format conversion is similar to CTB5.

Table 6 shows the coverage rates of training/testing words in our NLMWE word clustering dictionary (with 1,000 word classes and 4,999,890 unique words) and OOV rates in the testing sets of the Query treebank, CTB5 and CTB7.

As shown in Table 7, we manually construct a mapping from POS-tags to CPOS-tags and then apply this

[3] http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2005T01

[4] http://stp.lingfil.uu.se/ñivre/research/Penn2Malt.html

[5] http://stp.lingfil.uu.se/ñivre/research/chn_headrules.txt

[6] http://ilk.uvt.nl/conll/#dataformat

[7] http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2010T07

	Train	Test	
	NLMWE	NLMWE	OOV (in NLMWE)
Query	99.94	99.90	24.08 (99.58)
CTB5	91.60	91.60	6.25 (65.46)
CTB7	91.39	92.11	4.78 (57.03)

Table 6: Coverage rates (%) of training/testing words in NLMWE word clustering dictionary and OOV rates in the testing sets of the Query treebank, CTB5 and CTB7.

POS	CPOS
VA VC VE VV	1
NR NT NN	2
LC	3
PN	4
DT CD OD	5
M	6
AD	7
P	8
CC CS	9
DEC DEG DER DEV AS SP ETC MSP	10
IJ ON LB SB BA JJ FW	11
PU	12
X	13

Table 7: Mapping from POS-tags to CPOS-tags in CTB5 and CTB7.

mapping to CTB5 and CTB7 to obtain CPOS-tags for each word. For the Query treebank with POS-tags of the Peking University Corpus standard, we simply choose the first letter of the POS-tag to be its CPOS-tag.

4.2 Comparison of number of word classes

	wc.1000	wc.500	wc.100
nlmwe	87.13 (84.00)	86.21 (82.32)	85.86 (81.89)
+pos	88.70 (85.89)	87.84 (85.47)	87.28 (83.58)
+cpos	88.60 (85.68)	87.08 (83.79)	88.19 (84.42)
+pos+cpos	89.56 (86.74)	88.24 (85.05)	88.09 (84.84)

Table 8: The influence of the number of word classes to UAS$-p$ (%) of words and OOV words (numbers in the brackets) in the Query treebank.

Table 8 shows the influence of the number of word classes (of 100, 500, and 1,000) to UAS$-p$ (without punctuations kept and a beam size of 10) in the Query treebank. We observe that, for NLMWE, POS+NLMWE and POS+CPOS+NLMWE, UAS increases as the number of word classes increases. Here, 1,000 word classes performs the best accuracies.

In particular, in order to investigate the generalization of unknown words, we individually show the UAS of OOV words in the test set as influenced by the number of words classes (UASs in the brackets in Table 8). The absolute value of OOV words' UAS is lower than UAS of all words. Again, the combination of CPOS-tags, POS-tags, and NLMWE clustering with 1,000 classes yields the best result.

4.3 Comparison of POS, CPOS, NLMWE and their combinations

	UAS	UASoov	UASin
words	84.03	80.42	85.18
nlmwe	87.13	84.00	88.12
pos	87.18	85.05	87.85
pos+nlmwe	88.70	85.89	89.59
cpos	86.67	84.21	87.45
cpos+nlmwe	88.60	85.68	89.52
pos+cpos	87.63	85.05	88.45
pos+cpos+nlmwe	**89.56**	**86.74**	**90.45**

Table 9: The UAS$-p$ (%) scores of using feature templates that are related to POS, CPOS, NLMWE and their combinations in the Query treebank.

Table 9 intuitively shows the UAS scores of using feature templates that are related to POS, CPOS, NLMWE (of 1,000 word classes) and their combinations in the Query treebank. We use "words" to represent the feature templates listed in Table 1, i.e., no additional word generalized feature templates are used. For "pos", "cpos", and "nlmwe", they separately use not only the feature templates listed in Table 2 but also the word-related feature templates listed in Table 1. We observe that:

- when only using surface word related feature templates, the UAS (80.42%) of OOV words is the worst. This tells us that word classes are important for generalization. Referring to Table 1, we argue n-gram features such as $N_0wN_1wN_2w$ and their combinations with dependency labels such as N_0wN_0l contribute for the 80.42% accuracy of UAS;
- respectively appending POS/CPOS/NLMWE related features (Table 2) to words yields significant improvements of UAS;
- the combination of either two of POS, CPOS,

and NLMWE yields a better UAS than the individuals;

- finally, the UAS scores are the best for both OOV words (UASoov) and non-OOV words (UASin) when we combine POS, CPOS and NLMWE together.

	UAS	UASoov	UASin
words	71.93	66.98	72.32
nlmwe	76.98	71.88	77.39
pos	86.49	84.87	86.62
pos+nlmwe	86.41	84.91	86.53
cpos	84.43	82.46	84.59
cpos+nlmwe	84.82	83.64	84.92
pos+cpos	86.56	84.78	86.70
pos+cpos+nlmwe	**86.58**	**84.87**	**86.72**

Table 10: The UAS$-p$ (%) scores of using feature templates that are related to POS, CPOS, NLMWE and their combinations in CTB5.

	UAS	UASoov	UASin
words	72.71	68.80	72.94
nlmwe	78.03	75.17	78.19
pos	86.47	86.25	86.49
pos+nlmwe	87.01	86.69	87.03
cpos	84.21	83.04	84.28
cpos+nlmwe	85.03	83.81	85.10
pos+cpos	86.81	86.45	86.83
pos+cpos+nlmwe	**87.05**	**86.48**	**87.09**

Table 11: The UAS$-p$ (%) scores of using feature templates that are related to POS, CPOS, NLMWE and their combinations in CTB7.

	UAS	UASoov	UASin
words	69.76	64.16	70.09
nlmwe	74.56	69.86	74.83
pos	86.18	83.89	86.31
pos+nlmwe	86.36	84.38	86.48
cpos	83.66	81.94	83.76
cpos+nlmwe	84.55	82.35	84.68
pos+cpos	86.53	84.71	86.64
pos+cpos+nlmwe	**86.80**	**84.99**	**86.91**

Table 12: The UAS$-p$ (%) scores of using feature templates that are related to POS, CPOS, NLMWE and their combinations in CTB7 with Stanford dependency.

Table 10, 11 and 12 intuitively shows the UAS scores of using feature templates that are related to POS, CPOS, NLMWE (of 1,000 word classes) and their combinations in CTB5, CTB7 and CTB7 with Stanford dependency, respectively. Different with the 24.08% OOV rate of the test of the Query treebank, 6.25%/4.78% OOV rates of the test sets of CTB5/CTB7 are smaller. This makes UAS and UASin in these three tables quite close with each other. For NLMWE, we have the following observations:

- in CTB5, NLMWE does not bring a significant improvements after being appended to POS and/or CPOS related feature templates;

- in CTB7, appending NLMWE to POS yields an UAS$-p$ improvement of 0.54% (87.01% - 86.47%), which is significant; appending NLMWE to POS+CPOS further yields an UAS$-p$ improvement of 0.24% (87.05% - 86.81%);

- in CTB7 with Stanford dependency, appending NLMWE to POS yields an UAS$-p$ improvement of 0.18% (86.36% - 86.18%); appending NLMWE to POS+CPOS further yields an UAS$-p$ improvement of 0.27% (86.80% - 86.53%), which is significant;

- in CTB5, CTB7 and CTB7 with Stanford dependency, POS+CPOS+NLMWE yields the best UAS$-p$s.

Recall that we introduce CPOS to generalize POS in a sense. For example, all verb related POS tags of VA, VC, VE, and VV are taken as one CPOS tag. We hope that CPOS tags can capture the dependency grammar in a more generalized level than POS tags. For CPOS, we have the following observations:

- in CTB5, appending CPOS to POS yields a slight UAS$-p$ improvement of 0.07% (86.56% - 86.49%); appending CPOS to POS+NLMWE further yields an UAS$-p$ improvement of 0.17% (86.58% - 86.41%);

- in CTB7, appending CPOS to POS yields a slightly UAS$-p$ improvement of 0.34% (86.81% - 86.47%), which is significant; appending CPOS to POS+NLMWE further yields a slight UAS$-p$ improvement of 0.04% (87.05% - 87.01%);

- in CTB7 with Stanford dependency, appending CPOS to POS yields an UAS$-p$ improvement of 0.35% (86.53% - 86.18%), which is significant; appending CPOS to POS+NLMWE further yields an UAS$-p$ improvement of 0.44% (86.80% - 86.36%), which is also significant.

4.4 Comparison of golden/non-golden POS tags

	Not-Golden	Diff	Same	Golden
words	84.03	89.19	83.94	84.03
nlmwe	87.13	**91.89**	87.04	87.13
pos	87.13	89.19	87.09	87.18
pos+nlmwe	88.80	89.19	88.79	88.70
cpos	86.67	**91.89**	86.57	86.67
cpos+nlmwe	88.65	**91.89**	88.58	88.60
pos+cpos	87.53	89.19	87.50	87.63
pos+cpos+nlmwe	**89.56**	89.19	**89.57**	**89.56**

Table 13: The UAS$-p$ (%) of using the golden and not-golden POS-tags in the Query test set.

	Not-Golden	Diff	Same	Golden
words	71.91	63.75	72.14	71.93
nlmwe	76.96	**68.81**	77.19	76.98
pos	83.78	61.61	84.40	86.49
pos+nlmwe	83.94	62.55	84.54	86.41
cpos	82.80	61.78	83.39	84.43
cpos+nlmwe	83.17	62.90	83.74	84.82
pos+cpos	83.83	60.93	84.48	86.56
pos+cpos+nlmwe	**83.96**	61.95	**84.58**	**86.58**

Table 14: The UAS$-p$ (%) of using the golden and not-golden POS-tags in the CTB5 test set.

	Not-Golden	Diff	Same	Golden
words	72.71	65.46	72.87	72.71
nlmwe	78.03	**69.71**	78.22	78.03
pos	83.82	59.25	84.39	86.47
pos+nlmwe	84.41	59.82	84.97	87.01
cpos	82.61	60.71	83.11	84.21
cpos+nlmwe	83.42	62.23	83.91	85.03
pos+cpos	84.12	57.79	84.72	86.81
pos+cpos+nlmwe	**84.49**	59.13	**85.07**	**87.05**

Table 15: The UAS$-p$ (%) of using the golden and not-golden POS-tags in the CTB7 test set.

We finally replace the golden POS-tags in the test sets by POS-tags generated by a CRF-based POS-tagger. Table 13 (POS precision = 98.1%), 14 (POS precision = 97.7%), and 15 (POS precision = 98.1%) respectively show the UAS$-p$ differences of the Query, CTB5, and CTB7 test sets. In this three tables, "Not-Golden" and "Golden" (denoted as UASnot-golden and UASgolden, hereafter) stand for the UAS$-p$s of not using or using the golden POS-tags; "Diff" and "Same" (denoted as UASdiff and UASsame, hereafter) are the UAS$-p$s of those words whose POS-tags are different or similar with the golden POS-tags.

Specially, it is important for us to check whether NLMWE performs better when POS-tags are wrongly annotated. In all these three test sets, NLMWE's UASdiff performs better than that of POS or CPOS. Also, by combining wrong POS-tags (and/or CPOS-tags) and NLMWE together, UASdiff drops. Most importantly, when we look at UASdiff and compare POS with POS+NLMWE, NLMWE does supply POS (and CPOS). For example, in the CTB5 test set, UASdiff significantly increases from 61.61% to 62.55%; in the CTB7 test set, UASdiff significantly increases from 59.25% to 59.82%. On the other hand, by combining correct POS-tags (and/or CPOS-tags) and NLMWE together, UASsame increases, showing that NLMWE even being a good supplement for correct POS-tags.

Finally, when we check the results of appending NLMWE to POS/CPOS in the combination of UASsame and UASdiff, i.e., UASnot-golden, the increasing of UAS is significant in the Query test set (e.g., from 87.53% of POS+CPOS to 89.56% of POS+CPOS+NLMWE). In CTB5 and CTB7 test sets, we observe that NLMWE does bring increasings yet the increasings are not that significant. Referring back to Table 6, we argue that the low coverage (91-92% which is much lower than 99.9% in the Query treebank) of NLMWE clustering dictionary in the CTB5 and CTB7 response for this tendency.

4.5 Comparison to state-of-the-art

Table 16 shows the comparison of our system with state-of-the-art results on CTB5 and CTB7 test sets. We set the beam size of all our system variants to be 10. Note that most related researches only report their results on the CTB5 test set which makes direct comparison in the CTB7 test set unavailable. In the CTB5 test set, we observe that our final result of using POS+CPOS+NLMWE is comparable to most of related researches.

Besides the original data of CTB5 and CTB7, fol-

System	UAS$_{+p}$	UAS$_{-p}$
	CTB5	
Ours (words)	69.85	71.93
Ours (nlmwe)	75.05	76.98
Ours (pos)	84.65	86.49
Ours (pos+nlmwe)	84.52	86.41
Ours (cpos)	82.50	84.43
Ours (cpos+nlmwe)	82.91	84.82
Ours (pos+cpos)	84.72	86.56
Ours (pos+cpos+nlmwe)	84.70	86.58
Ours (pos+cpos+nlmwe)*	84.62	86.59
Ours (pos+cpos+nlmwe)$^{*,+s1s2}$	**84.88**	**86.79**
(Huang and Sagae, 2010)*	NA	85.2
(Zhang and Nivre, 2011)*	NA	86.0
(Zhang and McDonald, 2012)*	NA	**86.87**
(Li et al., 2012)*	NA	86.55
(Sun and Wan, 2013)	84.65	NA
(Hayashi et al., 2013)*	NA	85.9
(Ma et al., 2013)*	NA	86.33
	CTB7	
Ours (words)	70.57	72.71
Ours (nlmwe)	75.90	78.03
Ours (pos)	84.35	86.47
Ours (pos+nlmwe)	84.85	87.01
Ours (cpos)	82.08	84.21
Ours (cpos+nlmwe)	82.91	85.03
Ours (pos+cpos)	84.66	86.81
Ours (pos+cpos+nlmwe)	**84.89**	**87.05**
Ours (pos+cpos+nlmwe)*	84.84	86.99
	CTB7, Stanford	
Ours (words)$^+$	68.14	69.76
Ours (nlmwe)$^+$	73.00	74.56
Ours (pos)$^+$	84.80	86.18
Ours (pos+nlmwe)$^+$	84.97	86.36
Ours (cpos)$^+$	82.27	83.66
Ours (cpos+nlmwe)$^+$	83.19	84.55
Ours (pos+cpos)$^+$	85.11	86.53
Ours (pos+cpos+nlmwe)$^+$	**85.43**	**86.80**
MaltParser (libsvm)$^+$	NA	78.0
MSTParser (1st-order)$^+$	NA	78.9
Mate (2nd-order)$^+$	NA	**83.1**

Table 16: Comparison of our system with state-of-the-art results on CTB5/CTB7 test sets. Here, UAS$+p$/$-p$ stands for UAS (%) with/without punctuations taken into computing. * stands for using (Zhang and Clark, 2008)'s head rules. $^+$ stands for using (Chang et al., 2009)'s head rules. $^{+s1s2}$ stands for appending S_1 and S_2 related feature templates.

lowing (Che et al., 2012), we further apply the Stanford Chinese Dependency (Chang et al., 2009) to the CTB7's train/dev/test sets and compare the UASs with well known open-source systems. Different from the head rules defined in (Zhang and Clark, 2008) which yield 12 dependency labels, the Stanford Chinese Dependency has its own head rules as described in (Chang et al., 2009) which yields 46 fine-grained dependency labels. We directly used the UASs of these systems reported in (Che et al., 2012). In this test set, we observe that our system is significantly better than open-source systems of MSTParser of version 0.5[8] (McDonald and Pereira, 2006), Mate parser of version 2.0[9] (Bohnet, 2010) (2nd-order MSTParser) and MaltParser of version 1.6.1 with the Arc-Eager algorithm[10] (Nivre et al., 2006).

5 Conclusion

We have investigated the influence of generalization of words to the final accuracies of Chinese shift-reduce dependency parsing. We designed feature templates by making use of words, POS-tags, CPOS-tags, NLMWE-based word classes and their combinations. NLMWE-based word classes is shown to be an important supplement of POS-tags, especially for the automatically generated POS-tags. Experiments on a Query treebank, CTB5 and CTB7 show that the combinations of features from CPOS-tags, POS-tags, and NLP-WE-based word classes yield the best UASs. Our final UAS$-p$ of 86.79% on the CTB5 test set is comparable to state-of-the-art results. Our final UAS$-p$ of 86.80% and 87.05% on the CTB7 Stanford dependency test set and original test set is significantly better than three well known open-source dependency parsers.

Acknowledgments

The authors thank Wanxiang Che for processing the CTB7 data into Stanford dependency format, and the anonymous reviewers for their feedback that substantially improved this paper.

References

Yushua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Bernd Bohnet. 2010. Top accuracy and fast dependency

[8] http://sourceforge.net/projects/mstparser
[9] http://code.google.com/p/mate-tools
[10] http://www.maltparser.org/

parsing is not a contradiction. In *Proceedings of COLING*, pages 89–97, Beijing, China, August. Coling 2010 Organizing Committee.

Pi-Chuan Chang, Huihsin Tseng, Dan Jurafsky, and Christopher D. Manning. 2009. Discriminative reordering with Chinese grammatical relations features. In *Proceedings of SSST-3*, pages 51–59, Boulder, Colorado, June. Association for Computational Linguistics.

Wanxiang Che, Valentin Spitkovsky, and Ting Liu. 2012. A comparison of chinese parsers for stanford dependencies. In *Proceedings of ACL (Volume 2: Short Papers)*, pages 11–16, Jeju Island, Korea, July. Association for Computational Linguistics.

Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. 2013. Efficient stacked dependency parsing by forest reranking. *Transactions of the Association for Computational Linguistics*, 1(1):139–150.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL*, pages 1077–1086, Uppsala, Sweden, July. Association for Computational Linguistics.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Columbus, Ohio, June. Association for Computational Linguistics.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of Co-NLL*, pages 63–69.

Zhenghua Li, Min Zhang, Wanxiang Che, and Ting Liu. 2012. A separately passive-aggressive training algorithm for joint POS tagging and dependency parsing. In *Proceedings of COLING 2012*, pages 1681–1698, Mumbai, India, December. The COLING 2012 Organizing Committee.

Ji Ma, Jingbo Zhu, Tong Xiao, and Nan Yang. 2013. Easy-first pos tagging and dependency parsing with beam search. In *Proceedings of ACL (Volume 2: Short Papers)*, pages 110–114, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88.

Joakim Nivre, Johan Hall, , and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, pages 2216–2219.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of IWPT*, pages 149–160.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Weiwei Sun and Xiaojun Wan. 2013. Data-driven, pcfg-based and pseudo-pcfg-based models for chinese dependency parsing. *Transactions of the Association for Computational Linguistics*, 1(1):301–314.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*, pages 384–394, Uppsala, Sweden, July. Association for Computational Linguistics.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. 2007. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Mo Yu, Tiejun Zhao, Daxiang Dong, Hao Tian, and Dianhai Yu. 2013. Compound embedding features for semi-supervised learning. In *Proceedings of NAACL-HLT*, pages 563–568, Atlanta, Georgia, June. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of EMNLP*, pages 562–571, Honolulu, Hawaii, October. Association for Computational Linguistics.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of EMNLP-CoNLL*, pages 320–331, Jeju Island, Korea, July. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of ACL:HLT*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.