Dictionary Interface for Neural Word Segmentation

Yuezhou Sun
University of California, San Diego

Abstract

Recently, neural networks using deep learning techniques achieved great success on Chinese Word Segmentation. However, traditional methods based on the maximum matching algorithm maintain the advantage that they can easily update a dictionary to accommodate for rare or novel words. In this paper, I propose a plug-in dictionary interface with a set of training strategies to help neural networks make reference to a dynamically changing dictionary. Empirical results and statistical analyses show that my method reduces the domain transfer challenge induced by out-of-vocabulary words when appropriate dictionaries are provided.

1 Introduction

The Chinese Writing System represents a sentence as a continuous string of characters without visible markers of word boundaries. However, word segmentation is an important first step to various natural language processing problems and an interesting task for language models to complete, because the resolving of word boundaries require complex semantic and syntactic knowledge (Xue, 2003). The effort to develop better algorithms for automatic Chinese Word Segmentation (CWS) dates back to the late 20th century (Zhao et al., 2019). Although in recent years, neural networks using deep learning have achieved remarkable success on the CWS task, the traditional dictionary-based methods using the maximum matching algorithm maintain the advantage of flexibility that they can relatively easily handle rare or novel words by updating their dictionary.

In this paper, I propose to implant this dictionary lookup ability into neural networks using a plug-in dictionary interface and a set of training strategies. My method has three privileges. First, it can be applied on top of any model without modification to the network structure. Second, the dictionary is an interface instead of an integral part of the model, so its removal has little effect on the model's original performance. Third, the interface allows a neural network to dynamically make reference to different dictionaries in the test stage without further training. I will demonstrate with empirical results and statistical analyses that a model trained under my method would benefit greatly from an appropriate dictionary at test stage.

2 Background

Traditional CWS algorithms, which adopt the maximum matching algorithm, highly rely on dictionaries (Xue, 2003). The algorithm searches through all potential segmentations of the target sentence to maximize the number and lengths of words that match with some entry in a pre-defined dictionary. Although purely statistical approaches that calculate the mutual information between adjacent characters exist (Sproat and Shih, 1990), they do not perform well on their own. Instead, the statistical information such as word frequency or morpheme distribution typically get combined with dictionary-based methods to obtain the best performance (Sproat and Shih, 1996). The two existing CWS toolkits, Jieba and CKIP (Ma and Chen, 2005), fall into this category. Both use the max matching algorithm to find valid words, apply heuristic rules referring to statistical and manually defined linguistic features to resolve ambiguity, and extract Out-Of-Vocabulary words (OOV) using the character distribution.

In 2003, Xue proposed to reformulate CWS as a tagging problem, where the goal is to label each character with its position within the word. The view became widely accepted, and researchers addressed it with some variation of conditional random fields (Peng et al., 2004; Andrew, 2006). This new problem setup also made the task directly suitable for neural networks when deep learning came into natural language processing. Neural networks incorporate contextual and hierarchical information from the input data using different network structures. Previous studies have experimented with CNN (Liu et al. 2018), bi-LSTM (Zhou et al., 2017; Ma et al., 2018; Chuang, 2019), and transformer (Wu et al., 2019) and achieved promising results. Word embedding that automatically captures semantic and syntactic information also puts neural networks at a great advantage. A number of works pretrain word embeddings to distill linguistic knowledge, using skip-gram (Zhou et al., 2017; Ma et al., 2018), ELMo (Chuang, 2019), or BERT (Huang et al., 2019). With many of these models achieving promising results, deep-learning-based neural segmenters have been shown superior to traditional methods, at least in an experimental setting (Chuang, 2019).

However, neural word segmenters face a challenge of cross-domain transferability, which refers to the fact that many models' performance become compromised when they are tested on a different corpus. This is because that most neural networks are trained on a human annotated dataset in a supervised manner, and then tested on a small subset of the data which had been withheld from training, so the training set and test set are highly similar in terms of sentence structures and common vocabulary. In fact, Out-of-Vocabulary (OOV) words pose a major challenge at domain transfer, as widely recognized (Xue, 2003; Ma et al., 2018; Chuang, 2019) and later demonstrated in the Section 6.

3 Related Works

Although most models implicitly address the OOV word problem via optimizing the network structure to capture more morphological knowledge, some works also introduce dictionaries during training.

Liu et al. generated pseudo labeled data using an external dictionary to explicitly expose the model to rare words (2018). The same group used posterior regularization to increase the likelihood of dictionary entries in the model distribution (Liu et al., 2019). Although both works incorporate dictionaries, their main focus is to reduce the demand of human annotation. The dictionary is discarded after training and one must retrain or find-tune the model to modify the dictionary information.

Zhang et al. built feature vectors to encode whether a character forms words with its surrounding characters according to an external dictionary, which can be altered at the test time (2018). They demonstrated that the dictionary feature not only improves in-domain performance, but also helps with cross-domain adaptation when domain-specific lexicons get added into the dictionary. This work is highly relevant to mine, since I also use a dictionary to build dictionary vectors as model input. Nonetheless, our methods differ in multiple perspectives.

To begin with, Zhang et al. designed a network to incorporate dictionary vectors, using two bidirectional Long Short-Term Memory networks (bi-LSTM) to process the sentence input and dictionary vectors semi-independently from each other. My method concatenates the dictionary vector into each character's embedding as part of its feature, and therefore can be built on top of any network without modifications to the structure. Zhang et al. used conditional random fields (CRF) to make the final prediction, while mine uses argmax from SoftMax for speed and simplicity, although CRF can be added to further improve the results. Most importantly, Zhang et al. used an external dictionary during training to augment their model's vocabulary knowledge. The dictionary is kept during testing and therefore function as an integral part of the model. In my study, the training dictionary is directly collected from the training corpus, so it does not provide further information to the model and can be discarded during test. What it gives the model instead is the ability to access customized dictionaries without additional training. Therefore, my work can be viewed as a reproduction and complementation of Zhang et al. 2018.

4 Model Description

My model is a simple bidirectional Long Short-Term Memory neural network (bi-LSTM) following the success of previous works (Zhou et al., 2017; Ma et al., 2018; Chuang et al., 2019).

4.1 Sentence Embedding

The sentence input is embedded as character unigram and bigram vectors using an embedding lookup layer. The inclusion of bigram information is found crucial to the model's performance. Dropout is added on the embedded sentence to promote robustness.

4.2 Dictionary Vectors

Given a model hyperparameter of window size w and an input sentence $x = (x_1, x_2, \ldots, x_n)$, I calculate a dictionary vector d_i of dimension $2w+1$ for each character x_i, so that $d_i = \{d_i^{i-w}, d_i^{i-w+1}, \ldots, d_i^{i}, \ldots, d_i^{i+w-1}, d_i^{i+w}\}$. Each element d_i^j in the dictionary vector is a one-hot feature representing whether the character substring $(x_i, x_{i+1}, \ldots, x_j)$ or $(x_j, x_{j+1}, \ldots, x_i)$, depending on which index is greater, is a valid dictionary entry. If an index exceeds the sentence length thus making the substring span invalid, the element takes value of zero. If there is no dictionary provided, the entire vector takes value of zeros.

The dictionary vector of each character in the sentence is concatenated with the unigram and bigram embeddings, so that x_i is represented as $\{e_i, e_{i-1\_i}, d_i\}$ when fed into the neural network.

4.3 The Bi-LSTM Layer

A Bi-LSTM consists of a forward LSTM and a backward LSTM, which process a same sequence forward or backward respectively. While there exists debates on whether stacked or non-stacking bi-LSTMs works better for CWS (Ma et al., 2018; Zhang et al., 2018; Chuang 2019), I empirically find the non-stacking version to converge faster and achieve better results. I concatenate the hidden state at each character position from both LSTMs for the layer output. Again, recurrent dropout is added to avoid over-fitting.

4.4 BIES Classification

The Bi-LSTM output of each character is passed through a linear transformation layer followed by the SoftMax function. This final classification layer scores the character for each of the four tags, including (B)eginning of word, (I)ntermediate of word, (E)nd of word, and (S)ingle-character word. The model's prediction gets compared with the ground truth labels and the Cross-Entropy Loss is applied.

5 Training Strategies

The purpose of adding dictionary vectors is to build a plug-in interface for external dictionaries on any neural network without detriments to the model's original task performance. That being said, the model is expected to make word segmentation decisions primarily on its own linguistic knowledge and only adjust its predictions if the dictionary vector suggests something special. To achieve this goal, I devise two training strategies. Ablation studies on these strategies will be provided in Section 6.6.

5.1 Dictionary Vector Dropout

I add random dropout to the dictionary vectors to avoid over reliance on dictionary inputs. The dropout is applied at high level during the construction of dictionary vectors, rather than using a regular dropout layer in the neural network. If a word is dropped in one training sample, it is not marked in the dictionary vectors of any involved characters in this sample. Although a regular dropout layer could achieve the same effect, I implement the dropout in this fashion to assist the following strategy.

5.2 Word Coining

Imagine during test, if the model fails to recognize an unknown word, we want to add it into the dictionary to make the model recognize it. Therefore, given a same input sentence but different dictionary vectors, a model should make different word boundary decisions. To explicitly teach this behavior to the model, I add this strategy named "word coining", where I randomly glue up two adjacent words to coin a "new word" and change the dictionary vectors and ground truth labels accordingly during training. The new words are dynamically coined for each training sample and never get dropped from the dictionary vectors. While the dictionary vector dropout strategy could potentially lead the model to downplay dictionary vectors, the word coining strategy reiterates their role of importance.

6 Experiments

6.1 Datasets

Dataset	Training Set	Test Set	Test OOV Rate
AS	709K Sentences / 8,368K Tokens	14K Sentences / 198K Tokens	4.30
CityU	53K Sentences / 2,403K Tokens	1K Sentences / 68K Tokens	6.44
MSR	92K Sentences / 4,051K Tokens	4K Sentences / 184K Tokens	2.65
PKU	47K Sentences / 1,826K Tokens	5K Sentences / 173K Tokens	5.75
UD	4K Sentences / 156K Tokens	500 Sentences / 20K Tokens	12.06
Weibo	20K Sentences / 689K Tokens	2K Sentences / 73K Tokens	6.80
ZX	2K Sentences / 97K Tokens	1K Sentences / 48K Tokens	6.51

Table 1: Datasets.

I trained my model on five general CWS datasets including Academia Sinica Taipei (AS), City University of Hong Kong (CU), Beijing University (PK), Microsoft Research Beijing (MSR), and Chinese Universal Treebank (UD). The first four are provided in the Second International Chinese Word Segmentation Bakeoff (Emerson, 2005) while the last is taken from the Universal Dependencies 2.0 and used as the Conll2017 shared task (Zeman et al., 2017). I additionally used two datasets of special domains for testing. The Chinese Weibo Dependency Treebank (Weibo) is collected by Wang et al. (2014) from the social media Sina Weibo, the Chinese equivalent to Twitter. Zhuxian (ZX) is a free Internet novel annotated by Liu and Zhang (2012).

To ensure consistency, I converted all datasets into simplified Chinese and full-width tokens. I also re-chunked samples that contained multiple sentences or independent clauses separated by semicolons and filtered out samples with only one character. Table 1 exhibits the statistics of each dataset after preprocessing. The test set OOV rate is calculated as the occurrence frequency of words in the test data that are not found in the training set.

6.2 Evaluation Metric

Following previous works, I use F-score as the accuracy metric. The score is computed as $f = 2 \times p \times r / (p+r)$ where precision p is the number of correctly segmented words divided by the total number of words in the model's predicted segmentation, and recall r is that divided by the total number of words in the ground truth segmentation.

6.3 Experiment Setup

Dictionary vector window size	7
Character unigram embedding size	64
Character bigram embedding size	32
LSTM hidden size	128
Character input dropout	0.5
Dictionary vector dropout	0.5
LSTM recurrent dropout	0.2
Optimizer	Adam
Learning rate	0.001
Training epochs	30

Table 2: Hyper-parameters.

On each dataset, I train a Bi-LSTM using dictionary vectors and my training strategies with the hyper parameters listed in Table 2 and compare it with a baseline using the same model structure but not dictionary vectors. I collect words with length greater than one from each training set to construct the training dictionary and I randomly sample 10% of the training data to build a validation set.

6.4 In-Domain Results

For in-domain testing, I build the test dictionary from the test data. Table 3 demonstrates each model tested on its test set under three conditions: without a dictionary (No Dict), with the training dictionary (Train Dict), and with the test dictionary (Test Dict).

Train Cond	Test Cond	AS	CityU	MSR	PKU	UD
Baseline	No Dict	**95.24**	95.45	97.02	95.04	92.47
Ours	No Dict	94.49 (-0.75)	95.02 (-0.40)	96.92 (-0.10)	94.90 (-0.14)	92.34 (-0.13)
	Train Dict	95.14 (-0.10)	**95.59 (+0.14)**	**97.48 (+0.46)**	**95.40 (+0.36)**	**93.02 (+0.55)**
	Test Dict	**97.82 (+2.58)**	**98.25 (+2.80)**	**98.80 (+1.78)**	**97.57 (+2.53)**	**98.16 (+5.69)**

Table 3: In-domain results.

As shown, my model performs on par with the baseline when it uses the same dictionary as in training to construct dictionary vectors. Meanwhile, its performance only drops slightly under the no dictionary condition. In both conditions, no additional information is provided to the model other than what it has seen during training. Therefore, my method does not negatively affect the model's original performance.

Under the Test Dict condition, where additional vocabulary knowledge is provided, my model significantly outperforms the baseline, indicating that the model has learned to make correct reference to the dictionary interface.

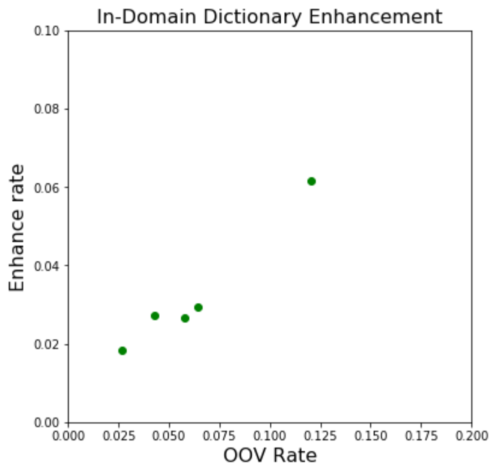

Figure 1: The in-domain enhance rate positively correlates with the test-to-train OOV rate.

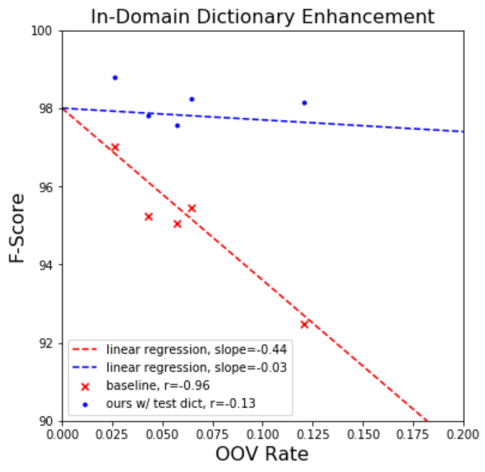

Figure 2: Test dictionary cancels out the negative correlation between a model's test performance and the test-to-train OOV rate.

If we take the Test Dict condition performance and the baseline performance to compute an enhance rate and plot it against the test-to-train OOV rate of each dataset, we get this nearly perfect linear relationship in Figure 1. The strong positive correlation between the OOV rate and the degree of a model's benefit from the test dictionary seems to imply that the performance improvement comes from the recognition of OOV words.

Figure 2 further illustrates this effect. In this scatter plot, the baseline models trained on each dataset exhibit a strong negative correlation between the test performance and the test set OOV rate. In a statistical analysis, the Pearson's correlation coefficient (r) between the two variables is as high as -0.96, confirming the fact that OOV words are one of the major threats to neural CWS methods. However, the r-score between the performance of my model exploiting the test dictionary and the OOV rate drops to -0.13. This is strong evidence to support that my method has specifically addressed and alleviated the OOV word problem.

6.5 Cross-Domain Results

For cross-domain testing, I take a model trained on one dataset and test it on another. I build the test dictionary from the training set of the target domain, so the dictionary does not reveal information about the test data itself, yet its vocabulary distribution better resembles the test set compared to the training dictionary. In this way, I am able to evaluate the effect of dictionary interface in a more realistic scenario.

		Train data	MSR					
Train cond	Test cond	Test data	AS	CityU	PKU	UD	Weibo	ZX
Baseline	No Dict		**82.15**	80.12	85.32	75.00	79.74	**77.86**
Ours	No Dict		81.39	79.61	85.25	75.04	79.36	77.16
	Train Dict		82.14	**80.20**	**85.57**	**75.33**	**80.03**	77.83
	Test Dict		**88.42**	**85.94**	**86.73**	**76.82**	**85.58**	**86.64**

Table 4: Cross domain results of models trained on the MSR dataset.

Table 4 demonstrates results of models trained on the MSR dataset. The complete table of all models tested on all datasets can be found in the Appendix. Similar to in-domain results, my models perform on par with the baseline under the No Dict and Train Dict conditions. They significantly outperform the baseline when the test dictionary is provided.

We plot the Test Dict condition enhance rate against the target-to-source domain OOV rate on figure 3 and again observe the strong linear relationship. In figure 4, the blue datapoints representing our models' performance under the Test Dict condition regress to an almost horizontal line. This time, the r score between the baseline F-score and the OOV rate is -0.62. However, it gets flattened to -0.01 after test dictionaries are provided. Although the cross-domain performances are not as high as in-domain results, the gap resides in factors other than the OOV challenge.

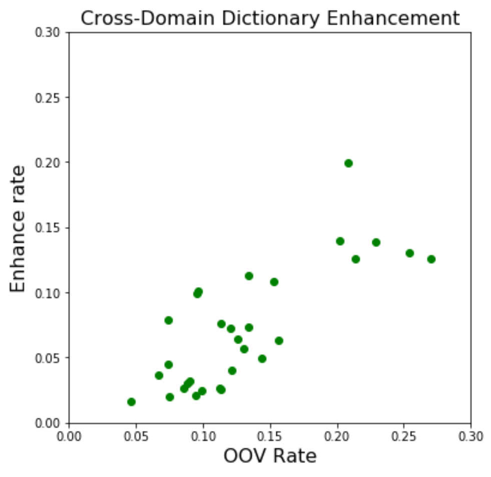

Figure 3: The cross-domain enhance rate also positively correlates with the target-to-source domain OOV rate.

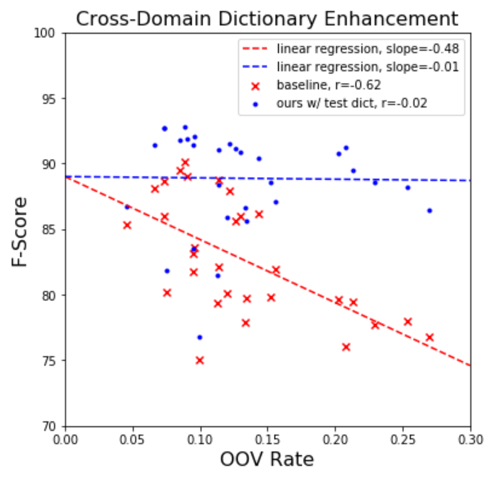

Figure 4: Test dictionary helps eliminate the effect of vocabulary differences across domains.

6.6 Ablation Study of Training Strategies

This section demonstrates the pivotal roles of my training strategies, including dictionary vector dropout and word coining. On top of the bi-LSTM baseline, I add dictionary vectors and my two training strategies step by step to train four models and evaluate them under all three test conditions. For brevity, I exemplify the results with models trained on the MSR dataset same as in the previous section. Complete records on all models can be found in the Appendix.

		Test Data						
Train Cond	Test Cond	AS	CityU	MSR	PKU	UD	Weibo	ZX
Baseline	no dict	82.15	80.12	97.02	85.32	75.00	79.74	77.86
+ Dictionary Vectors	no dict	39.88	36.90	36.25	37.09	40.14	37.60	54.01
	train dict	78.33	75.81	95.82	85.63	74.07	76.41	74.37
	test dict	90.36	88.01	99.54	91.44	79.55	86.07	88.34
+ Dictionary Vector Dropout	no dict	81.79	80.25	96.98	85.35	75.19	79.90	77.61
	train dict	82.41	80.48	97.52	85.65	75.40	80.37	78.05
	test dict	86.44	83.90	98.56	85.95	76.37	83.64	85.22
+ Word Coining	no dict	81.39	79.61	96.92	85.25	75.04	79.36	77.16
	train dict	82.14	80.20	97.48	85.57	75.33	80.03	77.83
	test dict	88.42	85.94	98.80	86.73	76.82	85.58	86.64

Table 5: Ablation study of training strategies on the MSR dataset.

As shown, adding dictionary vectors alone leads to heavy reliance on the dictionary information. Although the model achieves the highest score under the Test Dict condition, it does not appear acceptable without the test dictionary.

The dropout strategy restores the model's performance under the No Dict and Train Dict conditions back to the baseline level. However, the benefit it gains from the test dictionary gets constrained because the dropout to downplay of the dictionary vectors.

Finally, the word coining strategy strives to find a balance between the previous two. Its performance under the Test Dict condition is slightly worse than the dictionary-vector-only model, and its performance without test dictionaries is slightly compared to the dropout-only model. But overall, the combination of dictionary vectors with both strategies yields a satisfying result.

I have used the same dictionary dropout rate and word coining rate in my experiments. In practice, one can run a hyperparameter search to find the optimal balancing point between the two strategies.

6.7 Comparison with Related Works and Discussion

Because the test dictionary leaks information about the test data, my model is not directly comparable with the existent literature. However, I juxtapose my model with some representative works in the CWS field in Table 6 to give readers a more holistic picture.

Publications / Toolkit	AS	CityU	PKU	MSR	UD
Jieba	87.1	86.8	87.6	86.5	87.6
CKIP	97.7	94.3	93.9	92.0	91.2
Ma et al., 2018	96.2	97.2	96.1	98.1	96.9
Chuang et al., 2019	**98.0**	**98.6**	**97.7**	**98.7**	**98.3**
Wu et al., 2019	96.7	97.9	96.7	98.3	
Gong et al., 2019	95.2	96.2	96.2	97.8	
Huang et al., 2019	96.6	97.6	96.6	97.9	97.3
Liu et al., 2018	-	-	-	95.0	-
Zhang et al., 2018	95.9	96.3	96.5	97.8	-
This work (baseline)	95.2	95.5	95.0	97.0	92.5
This work (model)	97.8	98.3	97.6	**98.8**	98.2

Table 6: Comparisons.

Jieba and CKIP (Ma and Chen, 2005) are two popular toolkits using traditional algorithms. The performance was evaluated by Chuang et al. (2019).

The next three works investigate pretrained embeddings. Ma et al. used a stacked bi-LSTM with pretrained order-aware skip-gram embeddings. Chuang et al. is the state-of-the-art using a three-layer bi-LSTM with pretrained ELMo embeddings. Wu et al., used BERT with Glyce embeddings, a character embedding they proposed that is learned from Chinese character shapes.

The following two works explore multi-criteria training, where a single model deals with multiple datasets of different domains. Gong et al. introduced a switch-LSTM which simultaneously trains multiple LSTMs and maintains a high level "switch" state to route among them. Huang et al. trained a transformer on multiple datasets and used different output layers for each dataset.

The last two works, which have been introduced in Section 3, incorporate dictionaries in some manner. Liu et al. (2018) used an external dictionary to build pseudo labeled data. Zhang et al. employed two bi-LSTMs to semi-indenpendently process the sentence input and the dictionary features built from an external dictionary. Liu et al. (2019) was only tested on domain transfer and thus is not comparable with the rest of the works.

The next step of my study is two-fold. On one side, my dictionary vector and training strategies can be tested on a stronger baseline, some of which listed above. On the other side, although my method manages to bridge the gap of vocabulary differences across domains, other gaps such as the difference between sentence structures or even segmentation criteria remain. It will be interesting to combine my method with other transfer learning techniques.

7 Conclusion

In summary, I introduced a method to incorporate dictionary information into neural word segmenters using dictionary vectors and a set of training strategies, including dropout and word coining. My method enables a model to dynamically take in different dictionaries to recognize words unknown from the training data, without requirements for further training, modifications to the model's structure, or detriments to the model's original performance. Statistical analyses proved that my proposed dictionary interface greatly reduces the domain transfer challenge induced by OOV words.

References

Xue, Nianwen, and Libin Shen. "Chinese word segmentation as LMR tagging." *Proceedings of the second SIGHAN workshop on Chinese language processing-Volume 17*. Association for Computational Linguistics, 2003.

Zhao, Hai, et al. "Chinese Word Segmentation: Another Decade Review (2007-2017)." *arXiv preprint arXiv:1901.06079* (2019).

Sproat, Richard, and Chilin Shih. "A statistical method for finding word boundaries in Chinese text." *Computer Processing of Chinese and Oriental Languages* 4.4 (1990): 336-351.

Sproat, Richard, et al. "A stochastic finite-state word-segmentation algorithm for Chinese." *Computational linguistics* 22.3 (1996): 377-404.

Ma, Wei-Yun, and Keh-Jiann Chen. "Design of CKIP Chinese word segmentation system." *Chinese and Oriental Languages Information Processing Society* 14.3 (2005): 235-249.

Peng, Fuchun, Fangfang Feng, and Andrew McCallum. "Chinese segmentation and new word detection using conditional random fields." *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004.

Andrew, Galen. "A hybrid markov/semi-markov conditional random field for sequence segmentation." *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2006.

Liu, Junxin, et al. "Neural Chinese Word Segmentation with Dictionary Knowledge." *CCF International Conference on Natural Language Processing and Chinese Computing*. Springer, Cham, 2018.

Zhou, Hao, et al. "Word-context character embeddings for Chinese word segmentation." *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017.

Ma, Ji, Kuzman Ganchev, and David Weiss. "State-of-the-art Chinese word segmentation with bi-lstms." *arXiv preprint arXiv:1808.06511* (2018).

Chuang, Yung-Sung. "Robust Chinese Word Segmentation with Contextualized Word Representations." *arXiv preprint arXiv:1901.05816* (2019).

Wu, Wei, et al. "Glyce: Glyph-vectors for Chinese Character Representations." *arXiv preprint arXiv:1901.10125* (2019).

Huang, Weipeng, et al. "Toward Fast and Accurate Neural Chinese Word Segmentation with Multi-Criteria Learning." *arXiv preprint arXiv:1903.04190* (2019).

Liu, Junxin, et al. "Neural Chinese Word Segmentation with Lexicon and Unlabeled Data via Posterior Regularization." *The World Wide Web Conference*. ACM, 2019.

Zhang, Qi, Xiaoyu Liu, and Jinlan Fu. "Neural networks incorporating dictionaries for Chinese word segmentation." *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

Emerson, Thomas. "The second international Chinese word segmentation bakeoff." *Proceedings of the fourth SIGHAN workshop on Chinese language Processing*. 2005.

Zeman, Daniel, et al. "CoNLL 2017 shared task: multilingual parsing from raw text to universal dependencies." *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. 2017.

Wang, William Yang, et al. "Dependency parsing for weibo: An efficient probabilistic logic programming approach." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.

Liu, Yang, and Yue Zhang. "Unsupervised domain adaptation for joint segmentation and POS-tagging." *Proceedings of COLING 2012: Posters*. 2012.

Gong, Jingjing, et al. "Switch-lstms for multi-criteria chinese word segmentation." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019.

Appendix

Full experimental records.

Trained on AS

Train conditions	Test conditions	Test data						
		AS	CityU	MSR	PKU	UD	Weibo	ZX
no dict	no dict	82.15%	80.12%	97.02%	85.32%	75.00%	79.74%	77.86%
	train dict							
	test dict							
add dict	no dict	39.88%	36.90%	36.25%	37.09%	40.14%	37.60%	54.01%
	train dict	78.33%	75.81%	95.82%	85.63%	74.07%	76.41%	74.37%
	test dict	90.36%	88.01%	99.54%	91.44%	79.55%	86.07%	88.34%
add dropping	no dict	81.79%	80.25%	96.98%	85.35%	75.19%	79.90%	77.61%
	train dict	82.41%	80.48%	97.52%	85.65%	75.40%	80.37%	78.05%
	test dict	86.44%	83.90%	98.56%	85.95%	76.37%	83.64%	85.22%
add coining	no dict	81.39%	79.61%	96.92%	85.25%	75.04%	79.36%	77.16%
	train dict	82.14%	80.20%	97.48%	85.57%	75.33%	80.03%	77.83%
	test dict	88.42%	85.94%	98.80%	86.73%	76.82%	85.58%	86.64%

Trained on CityU

Train conditions	Test conditions	Test data						
		AS	CityU	MSR	PKU	UD	Weibo	ZX
no dict	no dict	90.09%	95.45%	83.60%	88.69%	81.79%	87.92%	81.98%
	train dict							
	test dict							
add dict	no dict	43.34%	39.62%	37.69%	39.50%	42.99%	39.96%	54.36%
	train dict	84.63%	89.60%	82.86%	87.25%	79.00%	81.56%	73.40%
	test dict	91.08%	99.22%	93.28%	92.03%	81.51%	87.06%	87.76%
add dropping	no dict	89.69%	95.10%	83.68%	88.70%	81.69%	87.28%	81.24%
	train dict	90.24%	95.69%	84.07%	89.14%	82.12%	87.81%	81.76%
	test dict	91.80%	98.31%	85.15%	89.71%	82.51%	89.64%	88.19%

add coining	no dict	89.74%	95.02%	83.71%	88.64%	81.73%	87.37%	81.15%
	train dict	90.16%	95.59%	84.03%	88.94%	81.99%	87.89%	81.58%
	test dict	92.77%	98.25%	92.07%	92.67%	83.53%	91.47%	87.13%

Trained on PKU

Train conditions	Test conditions	Test data						
		AS	CityU	MSR	PKU	UD	Weibo	ZX
no dict	no dict	85.61%	85.96%	85.97%	95.04%	79.37%	86.16%	79.84%
	train dict							
	test dict							
add dict	no dict	41.30%	37.68%	38.21%	39.64%	41.94%	38.46%	54.28%
	train dict	78.66%	77.82%	85.95%	93.23%	75.29%	78.47%	73.88%
	test dict	90.30%	88.43%	93.96%	99.41%	81.11%	86.66%	88.52%
add dropping	no dict	84.87%	84.97%	86.17%	94.86%	78.95%	84.94%	79.00%
	train dict	85.65%	85.78%	86.43%	95.38%	79.19%	85.69%	79.77%
	test dict	89.05%	89.05%	87.47%	96.97%	80.27%	88.96%	85.45%
add coining	no dict	85.46%	85.84%	86.10%	94.90%	79.27%	85.83%	79.55%
	train dict	86.08%	86.43%	86.37%	95.40%	79.90%	86.50%	80.19%
	test dict	91.12%	90.84%	92.73%	97.57%	81.47%	90.39%	88.52%

Trained on UD.

Train conditions	Test conditions	Test data						
		AS	CityU	MSR	PKU	UD	Weibo	ZX
no dict	no dict	79.49%	77.74%	76.09%	79.67%	92.47%	77.98%	76.75%
	train dict							
	test dict							
add dict	no dict	41.73%	37.62%	36.84%	37.50%	44.16%	38.27%	54.57%
	train dict	69.58%	67.48%	70.15%	72.13%	82.62%	66.91%	64.83%
	test dict	89.30%	87.60%	91.66%	90.12%	99.50%	85.88%	87.62%
add dropping	no dict	79.38%	77.60%	76.11%	79.61%	92.39%	77.70%	76.15%

	train dict	80.16%	78.53%	77.14%	80.66%	93.17%	78.76%	76.81%
	test dict	85.50%	83.55%	81.63%	84.48%	97.81%	83.72%	83.72%
add coining	no dict	79.09%	77.32%	75.79%	79.21%	92.34%	77.35%	76.55%
	train dict	79.83%	78.37%	77.06%	80.43%	93.02%	78.52%	76.85%
	test dict	89.48%	88.54%	91.27%	90.76%	98.16%	88.18%	86.40%

Trained on AS

Train conditions	Test conditions	Test data						
		AS	CityU	MSR	PKU	UD	Weibo	ZX
no dict	no dict	95.24%	89.46%	83.15%	88.13%	80.22%	88.78%	89.01%
	train dict							
	test dict							
add dict	no dict	41.33%	38.33%	35.74%	37.25%	42.16%	39.36%	55.41%
	train dict	92.18%	83.32%	81.28%	85.81%	78.97%	82.09%	83.45%
	test dict	98.51%	88.15%	90.01%	90.50%	81.14%	87.24%	88.45%
add dropping	no dict	94.69%	88.85%	82.93%	88.01%	80.47%	87.90%	88.00%
	train dict	95.37%	89.39%	83.16%	88.32%	80.49%	88.20%	88.32%
	test dict	97.45%	90.96%	85.26%	89.35%	81.18%	91.01%	91.46%
add coining	no dict	94.49%	88.38%	82.82%	87.88%	80.29%	86.98%	86.75%
	train dict	95.14%	88.69%	83.03%	88.20%	80.28%	87.21%	87.29%
	test dict	97.82%	91.80%	91.38%	91.37%	81.86%	91.02%	91.83%