# Outer Dictionary Network: An Efficient Network to Join Knowledge for Named Entity Recognition

## Abstract

In many AI applications, we need to extract some specific types of message from user's query, which is the main content of Named Entity Recognition (NER) task. In recent years, dictionary, one kind of outer knowledge, is proved to be important in NER tasks, and applied more and more constantly. Howerver, most of the current networks use it as word embedding vector, which will bring out such problems: a) out of vocabulary (OOV) problem; b) we need to retrain model when the list of words in dictionary is greatly changed. In this paper, we propose a new conception, called Outer Dictionary Network (ODN), which means to join the outer knowledge of dictionary into Neural Networks (NN), without using the specific word or word embedding vector. We prove in our paper that, the two mentioned problem will be solved in this way. We then propose a new structure, called Tag Embedding LSTM (TE-LSTM) to implement the ODN. Experiment results show that our model acheive a great improvement on several tasks within which dictionary is heavily depended on, with comparison with current state-of-the-art methods. And the network will change its output along with the content variation of dictionary, without training the model again.

## Introduction

Named Entity Recognition (NER) is a base task in Natural Language Processing (NLP) domain. Usually, we need to extract main information from users' sentences to help us understand their meanings, and then use the extracted information to do further process. For example, in weather domain, we may need to extract entities **date and position** to help us understand when and where the user is asked towards the weather; in music playing service, we may need to track entities **singer, song, style** to suit user's command; for many commom use, we may need entities **person, location, organization** to understand a sentence.

To accomplish this task, many method has been explored. Convolution Neural Network (CNN) is first used in Image Classification (Krizhevsky, Sutskever, and Geoffrey 2012), and then also proved powerful in Text Classification (Kim 2014). Therefore, CNN is also useful in NER tasks, whether to capture character-level features (Chiu and Nichols 2015; Ma and Hovy 2016; Peters et al. 2017), or to do word-level sequence labelling jobs (dos Santos, Xiang, and Zhou 2015; Strubell et al. 2017). To learn the correlation between words in sentence, Recurrent Neural Network (RNN) (Sutskever, Vinyals, and Le 2014) and Long Short-Term Memory (LSTM) (Sak, Senior, and Beaufays 2014) are proposed, and they achieve great success on machine translation and many other areas (Socher, Manning, and Ng 2010). They are then used in NER tasks and achieve significant improvement. These structures can be combined together and used in different situations (Yang, Liang, and Zhang 2018).

Among these years, many works have made innovative attempts on constributing new NN structures. These models, like Transformer structure (Vaswani et al. 2017) and its derivative methods, like OpenAI GPT (Radford et al. ), BERT (Devlin et al. 2018), XLNet (Yang et al. 2019), focus on base layer of NN structure, usually the embedding input layer, to get better sentence understanding. The output of these models can then be directly fed it or fine-tuned to do NLP tasks (Chen, Zhuo, and Wang 2019). Though will they make great improvement, it usually pays a long time to train these models. Through these works, we can conclude that, **The better and richer input knowledge there is, more efficient the model will be**.

Among NER tasks, there exists a little difference between different languages. In Chinese or other hieroglyphics language, a character is also a single word; and Chinese word is ofter referred to a character phrase which has length longer than one. In English or other letter-based language, character means a single letter, and word is a composition of letters; that is to say, English word is composed by characters, for which character level CNN can help improve NER tasks (Huang, Xu, and Yu 2015; Chiu and Nichols 2015; Ma and Hovy 2016; Peters et al. 2017; 2018; Yang, Liang, and Zhang 2018); but that can not be used in Chinese NER tasks, which is hieroglyphics, no letter is there in a word. What we call character embedding in Chinese NER tasks correspond to word embedding in English jobs; and the so-called word embedding in Chinese NER jobs is likened to word phrase embedding in English. For this reason, Chinese NER tasks is usually done by word baseline or character baseline structure, which has the different meaning with English NER tasks. Among Word baseline structures, we may use word segmentor to split sentence into word tokens, and then use word embedding to turn these token into vectors (Peng and Dredze 2015; 2016); while character structures simply treat each sentence as a sequence of Chinese characters, and then using corresponding character embedding to do NLP tasks (Zhang and Yang 2018; Gui et al. 2019). Furthermore, it seems useful when using word embedding within character baseline structure, or the oppisite, cause it can enrich the input knowledge. Recently, more and more Chinese datasets (Levow 2006; Peng and Dredze 2015; Zhang and Yang 2018) have been released to further research for Chinese NER research.

On the higher layer, most of the state-of-the-art NER methods use Conditional Random Fields (CRF) (Lafferty, McCallum, and Pereira 2001) to decode the sequence labelling result and compute loss. CRF can work even without any NN cells. Hand-crafted features already achieve high performance (Finkel, Grenager, and Manning 2005; Okazaki 2007) on many jobs.

Though with these powerful networks, word backgound knowledge, usually dictionary (same meaning to lexicon and gazette), is still necessary to do NER jobs in some situations. For example, in music playing service, “You raise me up” would be treated more likely as a chat intention rather than a song intention, unless the model is given that knowledge (see in Figure 1). Most of the state-of-the-art methods use word embedding (Pennington, Socher, and Manning 2014; Mikolov et al. 2017) to complete these tasks. But it then gives two problems:

- Out of vocabulary (OOV) problem. To our knowledge, word embedding will always lead to this problem. Furthermore, it is hard and burdersome to make embedding vector for every word in dictionary. As for entity **song** in music playing service, word embedding size will be too large; and sometimes will the length of song be too long to make word vector.
- Retraining problem. When the content of dictionary changes abundantly, the trained model would have poor performance doing the same job. Still use the entity **song** as an example, the dictionary would be changed monthly or even daily, which forces us to retrain the model synchronously.

Faced with these problems, we proposed a new conception, called Outer Dictionary Network (ODN), which means using the dictionary’s outer trait, without using its specific content. For example, we use the tag of word, like **song**, rather than a specific song name, to help model understand the outer knowledge. In this way, word embedding is no longer needed for Chinese NER task in ODN. Exprement results show that in dictionary depended-on task, ODN gives the best result among state-of-the-art methods. And when the content of dictionary changes, the model can synchronously update its result without retraining.

## Contributions

In summary, we make these contributions:

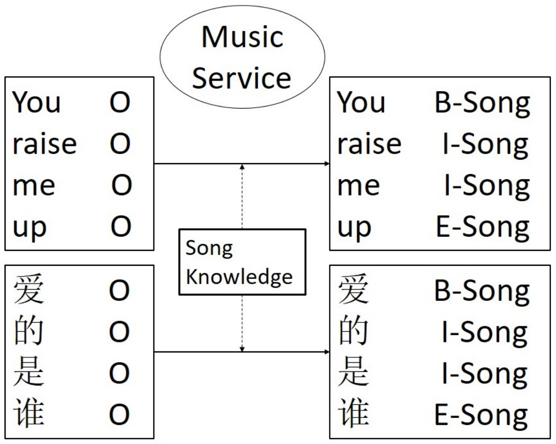

Figure 1: For NER situation in music service, dictionary is necessary to do NER task. Given the certain knowledge, we tend to denote the corresponding words as a song rather than others.

- We proposed a new conception: Outer Dictionary Network (ODN), which is used to update the content of dictionary without retraining the model.
- We proposed a new structure, Tag Embedding LSTM (TE-LSTM), to realize ODN. The networks use the word tag knowledge, rather than the word itself, to improve the performance of NER tasks.
- Experiment results have shown that, our model gives significant improvements on dictionary depended-on tasks among many state-of-the-art models.
- We release a new dataset in music service domain, for further scientific research.

## Related Work

Success of BERT (Devlin et al. 2018) and XLNet (Yang et al. 2019) show that input embedding knowledge is important and useful for many upper NLP tasks. More useful knowledge is there in the input vector, more efficiently the model would work. To a great degree, embedding layer decides the performance. Many useful features have been explored in NER jobs, such as character level message, word level message, position message, handwriting message. Dictionary knowledge is one of them. Especially in some NER task, like music service, dictionary is heavilly depended on to make better performance.

Most of the recent models use dictionary as word embedding input, or the word’s string format, to give the model related message. Lattice LSTM (Zhang and Yang 2018) use the dictionary message as additional input to character baseline LSTM, which can give the model more sentence knowledge. Lexicon Rethinking CNN (Gui et al. 2019), incorporate the dictionary knowledge into correlated CNN layer, and then the coflict word tokens could be solved by rethinking. Stanford NER (Finkel, Grenager, and Manning

2005) use the dictionary knowledge as string format feature, matched them in sentences, in CRF model.

However, in some pratical areas, the content of dictionary will change greatly and constantly. In this way, most of the recent model need to be retrained after every abundantly change. To solve this problem, we need to construct a model which can still be used after such change. We then come up with the idea of ODN, and use TE-LSTM to realize this idea. As a result, no more word embedding is needed, and model needs only to be trained once. What we need is only the word tag knowledge to help the model better understand user's intention.

## Outer Dictionary LSTM

Our framework is based on Tensorflow (Abadi et al. 2015). As we said before, character and word has the different meaning in Chinese and English tasks. We hold the following views towards these difference:

- Common characters in hieroglyphics language, can be exhaustive. For Chinese, there are no more than 100, 000 characters in simplified Chinese. Therefore, no OOV problem will there be when we use character embedding in Chinese NER task. For the associated case in English NER task, common used words can approximately be treated as limited.
- Words in hieroglyphics language, or phrases in English, will be uncountable, because the number of them is always too large, and there are still many new words or pharases are still created every day. Thus, OOV error can not be avoided in these case.

Based on that, we give our definition of ODN as, a network which satisfy these conditions:

- Only embedding vector or features, of which size is limited, is used. Any feature that would cause OOV will not be used.
- Only the outer knowledge of dictionary or knowledge graph, like tag or relation, is used. No specific word level feature is used.
- The dictionary used should be exhaustive among all time, or among a time range.

We call these networks ODN, no matter the specific method to realize it, CNNs or LSTM. In this way, only character embedding will be use in Chinese NER jobs, no word embedding or other features like bichar (Chen et al. 2015) and softword (Peng and Dredze 2016). For English, character-based feature and word embedding are all allowed. We denote a sentence as a sequence of characters: $s = (c_1, c_2, ..., c_n)$.

### Character Based LSTM

We use character embedding lookup table $e_c$ to turn character id to vector:

$$a_i = e_c(c_i) \tag{1}$$

Usually a dropout layer is connected with the embedding output, which is proved useful in NER jobs (Ma and Hovy 2016). After that, bi-directional LSTM (Bi-LSTM), the most common structure used in NER tasks, is applied to learn the character relationship within the sentence. We use the hidden vector for LSTM output, that is, for input: $x = (a_1, a_2, ..., a_n)$, we get output: $h = (h_1, h_2, ..., h_n)$, where $h_i$ is the concatenation of forward ($\overrightarrow{h_i}$) and backward ($\overleftarrow{h_i}$) LSTM hidden vector:

$$h_i = [\overrightarrow{h_i}; \overleftarrow{h_i}] \tag{2}$$

### Tag Embedding Model

As cases show in Figure 1, we need to use dictionary for certain NER tasks. However, the constant variation of dictionary may lead to constant retraining problem. Therefore, we may not use the specific word in each dictionary, instead, only the outer knowledge will be used, which is the main principle in ODN. Here, we implement our ODN through Tag Embedding (TE), which uses the tag embedding as the outer knowledge for dictionary. Cause we use LSTM structure to utilize tag embedding, we call our network TE-LSTM.

Specifically, we do this by two step:

a) Match sentence with the word in dictionary using certain outer model method.

b) Turn match result into embedding vector, used as the input of LSTM.

**Match Policy**. In this part, we need to match all the words in given dictionary occured in current sentence. We impletement this by using Aho-Corasick Automation Algorithm (Aho and Corasick 1975). In summary, we build a KMP (Knuth, Morris, and Pratt 1977) + Trie Tree (De La Briandais 1959) algorithm to accomplish this task. In methematical, given sentence $s = (c_1, c_2, ..., c_n)$, and a dictionary $d_i$, which is composed of word set $D_i = \{w_1, w_2, ..., w_V\}$. We denote the result of match policy as:

$$r_{d_i,j} = [(f_1, b_1), (f_2, b_2), ..., (f_k, b_k)] \tag{3}$$

Here, $f_i$ and $b_i$ denote the front and back position of each match word in original sentence. Let $W_{f_i,b_i}$ represent the substring of sentence $s$ with the rank $[f_i, b_i]$, then each $(f_i, b_i)$ tuple forms a word in $D_i$. Parameter $k$ means the total number of such word matched in $d_i$, which is uncertain. Cause there will be conflict intervals after AC algorithm is performed, we need to select the meaningful intervals, the footnote $j$ thus means that there will be many valid interval arrays. To format the input, we make the following policy to select our valid interval arrays:

- We choose certain number of valid interval arrays, which is called **tag blank number**, denoted as $w$. If there is not enough candidate interval arrays, the padding interval array will be use to fill out the left position.
- We choose the interval arrays which cover the most characters in sentence (words for English), and have no conflict intervals inside itself. And the more characters the interval array has, the prior it will be in embedding position.

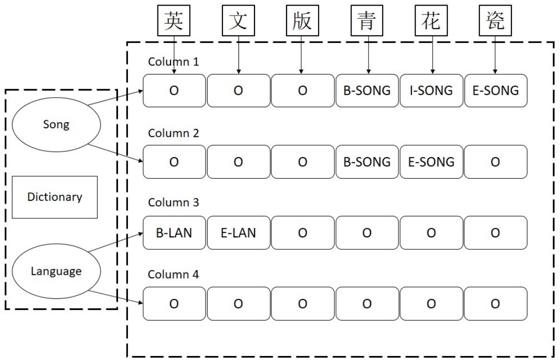

Figure 2: Illustration for tag embedding scheme. For dictionary “song”, there are two possible entities, the longer one is put in the first place (column 1). For dictionary “language” (abbreviated as “LAN”), only one possible entity will be matched, however in this case, since we set the tag blank number to 2, we should add one more padding blank to fill the position, which is showed as column 4.

**Tag Embedding Scheme**. After match policy is fullfilled, we need to turn that result into vector so that it can be understood by model. Cause the result of match policy is denoted as position, we can simply use it to get a new sequence form from original sentence. With dictionary $d_i$ and a matched word position rank range $[f_i, b_i]$, we then apply tag scheme for that range: turn the match word in that range to corresponding tag as shown in Figure 2, the unmatched part is turned to tag **O**.

Here, we choose **IOBES** scheme for it is proved better than IOB1 and IOB2 tag scheme in result by previous works (Chiu and Nichols 2015; Ma and Hovy 2016). Yet we need to emphasis that: the tag of dictionary here need not to be the same with the result tag species. As an illustration, to extract **organization** entity, we can use dictionary **organization suffix** to supply the suffix knowledge.

As related before, we choose **tag blank number**, $w$, of valid interval arrays and perform the tag scheme. We do the tag scheme in this way cause we think the position is important for model to understand, which here means to fix the certain tag result in appointed position. Of course, $w$ could be variable for different dictionaries, but that needs a bit more effort and seems low performance increase.

With the tag embedding sequence produced, denoting each tag sequence as: $s_t = (t_1, t_2, ..., t_n)$, we turn the result to vector using:

$$g_i = e_t(t_i) \tag{4}$$

where $e_t$ is the tag embedding lookup table. We use random uniform generated vector to setup this lookup table. Then the character embedding output $a_i$ and tag embeddding output $g_i$ is concatenated together:

$$x_i = (a_i; g_i) \tag{5}$$

Same to Character Based LSTM, a dropout layer is used on the concatenated result. Then the input is fed into the Bi-LSTM layer.

Table 1: Statistics of MusicNER dataset.

| Count | Sentences | Tokens | Entities |
|---|---|---|---|
| Train | 7,177 | 55,613 | 8,585 |
| Dev | 897 | 6,858 | 1,057 |
| Test | 897 | 6,775 | 1,047 |

## Loss and Training

CRF layer is applied on the top, same with many previous works (Collobert et al. 2011; Chiu and Nichols 2015; Ma and Hovy 2016; Yang, Liang, and Zhang 2018; Zhang and Yang 2018; Gui et al. 2019). Denote $y_j$ as a possible label sequence for sentence $s_i$ and $Y$ as the total set of all possible $y_j$, then the probability of $y_j$ is:

$$p(y_j|s_i) = softmax(\sum_i W_{c_i} * h_i + B_{c_i})_j \tag{6}$$

$W_{c_i}$, $B_{c_i}$ is respectively the emission matrix and transformation matrix correlate to $h_i$, and $j$ is the rank for $y_j$ in all possible label sequence set $Y$. While training, we take the $L_2$ regularization for all parameters into account together with CRF loss. With batch number $N$ and corresponding gold set $(s_i, y_i)$, the final loss can be computed as:

$$L = \sum_{i=1}^{N} log(p(y_i|s_i)) + \frac{\lambda}{2}||\theta||^2 \tag{7}$$

$\lambda$ is the $L_2$ hyper parameter, and $\theta$ stands for all the training parameters. At decoding step, we use the Viterbi algorithm (Viterbi 1967) to decode the result, finding the label sequence with the highest score:

$$y^* = \arg\max_{y_i \in Y} p(y_i|s_i) \tag{8}$$

## Experiments

### Settings

Here display the experiment settings within our work.

**Dataset**. For Chinese NER tasks in social domain, we explore the dataset, **Weibo NER** (Peng and Dredze 2015; He and Sun 2016) and **ResumeNER** (Zhang and Yang 2018), also used in Lattice LSTM (Zhang and Yang 2018) and Lexicon Rethinking CNN (Gui et al. 2019). In music domain, We use the dataset obtained from China Conference on Knowledge Graph and Semantic Computing (CCKS) 2018 (Liu et al. 2018). Call this dataset as **CCKS** dataset. Another music domain dataset is annotated by ourselves, and the tags are richer than CCKS. The statistics information of this dataset is listed in Table 1. We call it **MusicNER** dataset. For English NER task, we report the result on **CoNLL 2003** shared task (Sang and Meulder 2003).

**Data Preprocessing**. In our paper, all input file is turned to a certain format, which we call standard NER format. For example, “**Peter/B-PER Blackburn/E-PER**” is turned to string as {**"text": "Peter Blackburn", "slot": [0, 15, "PER"'**}. The numbers stand for the start position and end position in the original sentence. We do it this way since we can get the original input sentence without splitting sentence, and turn it into all other tag scheme (mainly IOB1, IOB2, IOBES) conveniently. Furthermore, we can easily statistics the sentence number and entity number in this format. On training, we turn the standard NER format into the tag scheme we appointed in hyperparameter file. In this way, take the CoNLL 2003 dataset as example, it will first be transformed into standard format, and then turned to IOBES tag scheme when training. In truth, this transformation doesn't change the content of dataset at all, but make it easier to process different format of dataset, and allow us to get the original input sentence to match the dictionary.

**Character Embedding**. In order to compare with Lattice LSTM (Zhang and Yang 2018) and Lexicon Rethinking CNN (Gui et al. 2019), we use the word vector provided in their papers (these two paper use the same character embedding and word embedding). There are character embedding and word embedding, we use the latter one as character embedding in our model, for we treat English word as a character in Chinese NER task, not a set of single English letters. For English NER task, we use Stanford's public available GloVe word embedding (Pennington, Socher, and Manning 2014), 100-dimensional, same with the BLSTM-CNNs-CRF (Ma and Hovy 2016).

**Tag Embedding**. As related before, we use uniform random policy to generate tag embedding lookup table. Given all tag number $l_t$ and tag embedding size $d_t$ as hyper parameter, we generate a matrix with shape $(l_t, d_t)$ for tag embedding lookup table.

**Dictionary Knowledge**. For Lattice LSTM and Lexicon Rethinking CNN, they use the same word embedding file to explore dictionary knowledge. For Stanford NER (Finkel, Grenager, and Manning 2005) model, which we will use in our experiment for comparison, we use different dictionaries for different dataset. And the same dictionaries will be use in TE-LSTM, the mehod we proposed. That is to say, for the same dataset, the dictionaries used in Stanford NER and TE-LSTM keep the same. We make up the dictionaries according to the follow principles:

- For exhaustive tag, we list all the items of that tag, extracted from **Knowledge Graph**, as a dictionary. For example, entity **Country** and **Project**.
- For tag which is limited but updated constantly, we list the current items of that tag, collected from **Knowledge Graph**, as a dictionary. For example, entity **Singer** and **Song**.
- For tag that is unlimited but has common prefix or suffix, we list all the common prefix or suffix as a dictionary. For example, entity **Organization** usually has a common suffix, such as: inc., company, league.
- For tag which is unlimited and seems no rule inside, we don't use dictionary for it.
- We setup the dictionaries mainly through **Knowledge Graph** we build, some of the dictionaries are enriched with the dataset.

Table 2: Hyperparameters

| Hyper-parameters | Value | Hyper-parameters | Value |
|---|---|---|---|
| char embeddding size | 50 | tag embedding size | 10 |
| lstm state size | 200 | tag blank number | [1,3] |
| lstm layers | 1 | batch size | [1,10] |
| dropout rate | (0.0, 0.5] | initial learning rate | [0.01,0.03] |
| decay rate | 0.05 | gradient clipping | 5.0 |
| momentum | 0.9 | L2 $\lambda$ | 1e-8 |

Comply with these principles, the dictionaries can be built and maintained for pratical use.

**Handcrafted Features**. Handcrafted features is only used in Stanford NER model. The specific features used in our experiment can be seen in our code. We also explored CRF-suite tools (Okazaki 2007) to test handcrafted features, but we finally choose Stanford NER for its convenience.

## Compared Models

For comparison, we impletement these models below:

**Lattice LSTM**. The author of Lattice LSTM (Zhang and Yang 2018) has generously provided his code on website. We directly use his code and settings, and run it on experiment dataset. We denote it as La-LSTM.

**Lexicon Rethinking CNN**. Same with Lattice LSTM, we use the code obtained from the author's website (Gui et al. 2019) and run it on our dataset. We denote it as LR-CNN.

**Character LSTM**. We impletement the character-level LSTM as the based model, no any other properties (like bichar, softword, position etc.) used. Denoted as C-LSTM.

**Stanford NER model**. Stanford NER tool (Finkel, Grenager, and Manning 2005) is used in our work to do comparision, for it is convenient to add handcrafted features and use dictionary (called gazette in its term) into model, which can be compared to our ODN intuitively. For short, denote it as ST-NER.

**Tag Embedding LSTM**. This is the model which we proposed in this paper. The character embedding is the same with C-LSTM, and the dictionary used is the same with ST-NER model for comparison. We denote it as TE-LSTM.

## Hyper-parameters and Strategies

Shown in Table 2, we choose the hyper-parameters referring to BLSTM-CNNs-CRF structure (Ma and Hovy 2016). Main training strategies we used are list as follow:

- **Fine-tuning**.We apply fine-tuning on all the embedding vector, which is proved useful by previous works (Collobert et al. 2011; Peng and Dredze 2015; Ma and Hovy 2016).
- **Dropout**. We choose the dropout rate between 0.0 and 0.5, for different dataset to get the best result.
- **SGD policy**. We use stochastic gradient descent (SGD) policy to train model, with momentum set to 0.9.

## Main Results

Compared to previous work, we report the best result of our experiment here.

Table 3: Main Results on Weibo NER.

| Model | NE | NM | Overall |
|---|---|---|---|
| La-LSTM | 53.04% | 62.25% | 58.79% |
| LR-CNN | **57.14**% | 66.67% | 59.92% |
| C-LSTM | 51.03% | 47.58% | 49.25% |
| ST-NER | 46.69% | 57.32% | 52.04% |
| TE-LSTM | 48.78% | **71.31**% | **60.10**% |

Table 4: Main Results on ResumeNER.

| Model | Precision | Recall | F1 |
|---|---|---|---|
| La-LSTM | 94.81% | 94.11% | 94.46% |
| LR-CNN | **95.37**% | 94.84% | 95.11% |
| C-LSTM | 94.31% | 94.27% | 94.29% |
| ST-NER | 94.15% | 93.74% | 93.94% |
| TE-LSTM | 95.32% | **95.15**% | **95.24**% |

Table 5: Main Results on CCKS.

| Model | Precision | Recall | F1 |
|---|---|---|---|
| La-LSTM | 87.34% | 75.03% | 80.72% |
| LR-CNN | 88.17% | 76.98% | 82.20% |
| C-LSTM | 75.66% | 71.12% | 73.32% |
| ST-NER | **91.00%** | 76.15% | 82.92% |
| TE-LSTM | 84.28% | **84.51%** | **84.40%** |

Table 6: Main Results on MusicNER.

| Model | Precision | Recall | F1 |
|---|---|---|---|
| La-LSTM | 86.24% | 84.43% | 85.33% |
| LR-CNN | 85.99% | 85.00% | 85.49% |
| C-LSTM | 84.11% | 82.50% | 83.30% |
| ST-NER | 88.00% | 85.48% | 86.72% |
| TE-LSTM | **93.09%** | **92.82%** | **92.96%** |

**Weibo NER Dataset**. As comparison to previous works, shown in Table 3, we report the F1-score for named entities (NE), nominal entities (NM) and overall entities. The dictionaries we use here are the set of **GPE_NAM**, **ORG_NOM**, **PER_NOM**, **LOC_NOM** and **Chinese Last Name**. We report the highest F1-score on this dataset, higher than LatticeLSTM and LR-CNN.

**ResumeNER Dataset**. Dictionary **Title Suffix**, **Organization Suffix**, **Project** and **China Peoples** is used here on ST-NER and TE-LSTM. Results on this dataset also support the efficiency of ODN.

**CCKS / MusicNER Dataset**. Cause we could only get the training dataset from CCKS website, we split that file into three part with portion 8:1:1 as train set, dev set and test set. Table 5 and 6 show the result of music domain dataset. The mainly dictionaries used here is **Singer** and **Song**. Results show that our model report the best result on both dataset on F1-score. With dictionary information given, we can see that ST-NER shows better performance than La-LSTM and LR-CNN. That, to some extent, cause by the lack of music domain knowledge in La-LSTM and LR-CNN. Compared to ST-NER, our model gives **1.48%** percent promotion of F1-score on **CCKS** dataset, and **6.24%** percent promotion on **MusicNER** dataset. The results prove that TE-LSTM is effective to utilize the knowledge of dictionary, though, without specially making word embedding vector for each word.

**CoNLL Dataset**. On this dataset, we list some of the effective model results reported by previous works. And we use dictionary **English Last Name**, **Location**, **Organization**, **MISC**. We make a suppose here that we can get all the real entities of location, organization and miscellaneous in English. With this prerequisite, we could see significant improvement on this dataset, 3.85% percent over BERT (Devlin et al. 2018) on F1-score, shown in Table 7. Though the result is amazing, **we need to emphasis that, it is hard and burdensome to collect all the entities in these dictionaries in reality**. For simplicity, we use the set of entities extracted from the dataset instead. So the result is only used to show that, ODN is powerful in using dictionary knowledge, but the way to collect dictionaries on this dataset is impratical, conflict with the principles we build dictionaries before.

## Analysis

As a further summarization for experiment results, we make the following conclusions:

**No OOV problem**. We don't need to consider OOV problem for the networks in ODN, for we only use character-based networks, and we only collect the dictionary which is exhaustive among all time or a certain time range. Furthermore, only the tag knowledge is used in ODN, not the specific word.

**Dictionary is important**. By comparing the results between C-LSTM and TE-LSTM, we would find great improvement after dictionary knowledge is given to model.

**Powerful in using dictionary knowledge**. Though without any handcrafted features, our model, TE-LSTM, performs better than ST-NER, with the same dictionary knowledge given.

**Helpful to Improve Recall**. All the experiment results on different dataset get the highest recall score, proving that ODN is a powerful tool on recall.

**More efficient when dictionary is more necessary**. By comparing the results on CCKS and MusicNER dataset to La-LSTM and LR-CNN, our model give significant improvement on these dataset, mainly because dictionary is necessary in music domain and, both La-LSTM and LR-CNN lack the knowledge of that. Results on CoNLL can also prove this view if the necessary dictionary can be built. On other datasets, the improvement is not so significant.

**Conflict Solving**. Though given dictionary knowledge, there exist many conflict items, labeled with different tag, to challenge the model. For example, "**French Open**", The word "**French**" will be given both the **S-MISC** tag and **B-ORG** tag here. TE-LSTM can solve these conflict cases well by using the dictionary knowledge together with the character embedding knowledge, as we could see in Table 7,

**Without Retraining**. We show it here that TE-LSTM need not to retrain the model when dictionary is greatly changed. The cost of retraining a model is changed to be

Table 7: Main Results on CoNLL 2003.

| Model | Precision | Recall | F1 |
|---|---|---|---|
| Ma and Hovy, 2016 | 91.35% | 91.06% | 91.21% |
| Yang et al. 2018 | — | — | 91.35% |
| Devlin et al. 2018 | — | — | 92.80% |
| C-LSTM | 87.32% | 85.25% | 86.27% |
| ST-NER | 86.02% | 84.63% | 85.32% |
| TE-LSTM | **96.63%** | **96.67%** | **96.65%** |

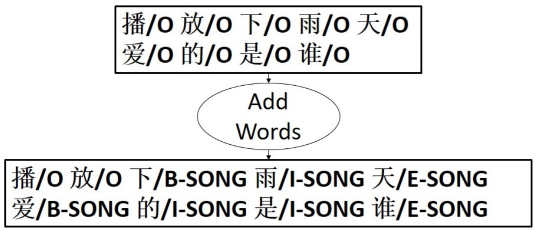
播/O 放/O 下/O 雨/O 天/O
爱/O 的/O 是/O 谁/O
Add Words
播/O 放/O 下/B-SONG 雨/I-SONG 天/E-SONG
爱/B-SONG 的/I-SONG 是/I-SONG 谁/E-SONG

Figure 3: In music service, after adding corresponding words into song dictionary, the new entities can be recognized by the model without retraining.

the cost of maintaining the dictionaries, which is more convenient and can be understood and modified by human.

Here, we use entity **Song**, in CCKS dataset, as example. Figure 3 shows the difference after adding corresponding words into dictionary **Song**, the model can then recognize these entities with the new given knowledge. Figure 4 shows the change after deleting redundant words in dictionary, the model will ignore these words after that operation. But there are also cases which will not change or partly change after deleting the words, shown in Figure 5, these cases exist because there are certain word phrases to discriminate the word as a song, or because part of the word is still in the dictionary. Retraining the model will make no sense on solving this problem usually.

## Conclusion

We proposed a new model to better explore the knowledge of dictionary without making embedding vector for each word, which helps us need not to retrain a model after the content of dictionary is greatly changed. And the results show that our model reach higher performance on dictionary-depended task, which certificated that our model is a powerful structure to utilize the knowledge of dictionary. And with ODN, the cost of training a new model is replaced by protecting the dictionaries, which is more understandable to human beings.

We have several directions to move ahead. For the structure itself, ODN can also be simply implemented using other structure, like the Lattice LSTM structure or CNNs structure, which also, is a powerful way to use dictionary knowledge. To solve the problem that, entity is still be recognized by the model after it is deleted from dictionary, we can construct a negative dictionary that contains list of words which we don't want it to be recognized. Furthermore, we can investigate networks that could search the needed knowledge

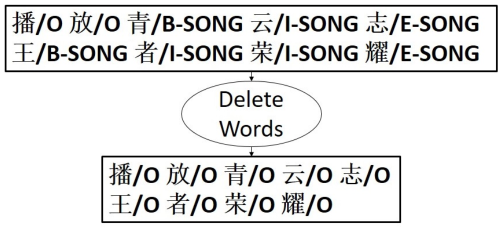
播/O 放/O 青/B-SONG 云/I-SONG 志/E-SONG
王/B-SONG 者/I-SONG 荣/I-SONG 耀/E-SONG
Delete Words
播/O 放/O 青/O 云/O 志/O
王/O 者/O 荣/O 耀/O

Figure 4: In music service, after delete corresponding words in dictionary, the corresponding entities can be ignored by the model without retraining.

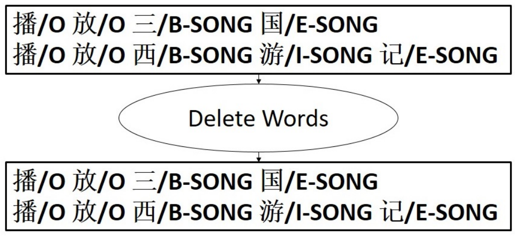
播/O 放/O 三/B-SONG 国/E-SONG
播/O 放/O 西/B-SONG 游/I-SONG 记/E-SONG
Delete Words
播/O 放/O 三/B-SONG 国/E-SONG
播/O 放/O 西/B-SONG 游/I-SONG 记/E-SONG

Figure 5: In music service, after delete corresponding words in dictionary, some of the the result will not change. Retraining will no help on these cases.

from website or knowledge graph by itself, which is more useful and promising in pratical.

## References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Aho, A. V., and Corasick, M. J. 1975. Efficient string matching: An aid to bibliographic search. *Commun. ACM* 18(6):333–340.

Chen, X.; Qiu, X.; Zhu, C.; Liu, P.; and Huang, X. 2015. Long short-term memory neural networks for Chinese word segmentation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1197–1206. Lisbon, Portugal: Association for Computational Linguistics.

Chen, Q.; Zhuo, Z.; and Wang, W. 2019. BERT for joint intent classification and slot filling. *CoRR* abs/1902.10909.

Chiu, J. P. C., and Nichols, E. 2015. Named entity recognition with bidirectional lstm-cnns. *CoRR* abs/1511.08308.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural language processing (almost) from scratch. *Machine Learning Research* 12 (2011):2493–2537.

De La Briandais, R. 1959. File searching using variable length keys. In *Papers Presented at the the March 3-5, 1959, Western*

*Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), 295–298. New York, NY, USA: ACM.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR* abs/1810.04805.

dos Santos, C.; Xiang, B.; and Zhou, B. 2015. Classifying relations by ranking with convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 626–634. Beijing, China: Association for Computational Linguistics.

Finkel, J. R.; Grenager, T.; and Manning, C. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, 363–370. Stroudsburg, PA, USA: Association for Computational Linguistics.

Gui, T.; Ma, R.; Zhang, Q.; Zhao, L.; Jiang, Y.-G.; and Huang, X. 2019. Cnn-based chinese ner with lexicon rethinking. 4982–4988. doi:10.24963/ijcai.2019/692.

He, H., and Sun, X. 2016. F-score driven max margin neural network for named entity recognition in chinese social media. *CoRR* abs/1611.04234.

Huang, Z.; Xu, W.; and Yu, K. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR* abs/1508.01991.

Kim, Y. 2014. Convolutional neural networks for sentence classification. arXiv:1408.5882.

Knuth, D.; Morris, Jr., J.; and Pratt, V. 1977. Fast pattern matching in strings. *SIAM Journal on Computing* 6(2):323–350.

Krizhevsky, A.; Sutskever, I.; and Geoffrey, E. H. 2012. Imagenet classification with deep convolutional neural networks. In *In Proceedings of Advances in Neural Information Processing Systems 25 (NIPS 2012)*.

Lafferty, J.; McCallum, A.; and Pereira, F. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001)*, 282–289.

Levow, G.-A. 2006. The third international Chinese language processing bakeoff: Word segmentation and named entity recognition. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, 108–117. Sydney, Australia: Association for Computational Linguistics.

Liu, K.; Guo, S.; Liu, S.; and Zhang, Y. 2018. China conference on knowledge graph and semantic computing (2018).

Ma, X., and Hovy, E. H. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR* abs/1603.01354.

Mikolov, T.; Grave, E.; Bojanowski, P.; Puhrsch, C.; and Joulin, A. 2017. Advances in pre-training distributed word representations. *CoRR* abs/1712.09405.

Okazaki, N. 2007. Crfsuite: a fast implementation of conditional random fields (crfs). http://www.chokkan.org/software/crfsuite/.

Peng, N., and Dredze, M. 2015. Named entity recognition for chinese social media with jointly trained embeddings. In *Processings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 548–554.

Peng, N., and Dredze, M. 2016. Improving named entity recognition for chinese social media with word segmentation representation learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 2, 149–155.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543.

Peters, M. E.; Ammar, W.; Bhagavatula, C.; and Power, R. 2017. Semi-supervised sequence tagging with bidirectional language models. *CoRR* abs/1705.00108.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. *CoRR* abs/1802.05365.

Radford, A.; Narasimhan, K.; Salimans, T.; and Sutskever, I. Improving language understanding by generative pre-training. Technical report, OpenAI.

Sak, H.; Senior, A.; and Beaufays, F. 2014. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. In *arXiv:1402.1128*.

Sang, E. F. T. K., and Meulder, F. D. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *CoRR* cs.CL/0306050.

Socher, R.; Manning, C. D.; and Ng, A. Y. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*.

Strubell, E.; Verga, P.; Belanger, D.; and McCallum, A. 2017. Fast and accurate entity recognition with iterated dilated convolutions. In *arXiv:1702.02098*.

Sutskever, I.; Vinyals, O.; and Le, Q. 2014. Sequence to sequence learning with neural networks. In *Proceedings of Advances in Neural Information Processing Systems 27 (NIPS 2014)*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. *CoRR* abs/1706.03762.

Viterbi, A. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* 13(2):260–269.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J. G.; Salakhutdinov, R.; and Le, Q. V. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR* abs/1906.08237.

Yang, J.; Liang, S.; and Zhang, Y. 2018. Design challenges and misconceptions in neural sequence labeling. *CoRR* abs/1806.04470.

Zhang, Y., and Yang, J. 2018. Chinese ner using lattice lstm. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*.