EXPLORING REPRESENTATION METHODS FOR SEQUENCE LABELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Sequence labeling is a general task encompassing a variety of applications in natural language processing, such as part-of-speech tagging and named entity recognition. Recent advances in representation learning can automatically encode word-level and character-level information. They allow neural networks to achieve the state-of-the-art without domain-specific feature engineering. However, the effectiveness of character-level representation modules is not clear, and how to leverage pre-trained embeddings have not been well studied. Therefore we first compare popular character-level representation modules with controlled experiments. From the results, we observed LSTM-based modules achieved better performance than CNN-based modules. Such comparison allows us to better understand existing representation modules and achieve further improvements. Also, we proposed a novel word-wise dropout strategy to carefully fine-tune pre-trained embeddings without shifting the whole semantic space. The resulting representation components help the sequence labeling model achieve the new state-of-the-art on three benchmark datasets. For further studies and improvements, we would release all implementations and codes to the public[1].

1 INTRODUCTION

Sequence labeling is a fundamental task in natural language processing (NLP). It has been applied to a variety of applications including part-of-speech (POS) tagging, noun phrase chunking, and named entity recognition (NER) (Ma & Hovy, 2016; Sha & Pereira, 2003). Typically, traditional methods relied on domain-specific feature engineering, and adapting them to new domains could be difficult (Lafferty et al., 2001; Sha & Pereira, 2003). Recent advances in representation learning break such limitation and make neural networks the state-of-the-art. Instead of handcrafting, these modules can automatically extract word-level and character-level information and have proven successful in many tasks (Reimers & Gurevych, 2017a; Conneau et al., 2017). In this paper, we try to find more effective representation modules and thus improving sequence labeling models.

However, few existing comparisons have been conducted among representation modules, and convincing improvements could only be made after thorough examinations of existing works. Actually, even the comparison between different sequence labeling frameworks is controversial. LSTM-CNN-CRF (Ma & Hovy, 2016) reported significant improvements over LSTM-CRF (Lample et al., 2016), but Reimers & Gurevych (2017b) ended up with an opposite observation.

Here, we first systematically compare and analyze the effectiveness of existing representation modules. To control experimental variates like the runtime environment and the pre-processing, we re-implemented LSTM-CRF and LSTM-CNN-CRF. From the results, we observed the long-short term memory (LSTM) network has more effectiveness than the convolutional neural network (CNN). Also, Liu et al. (2017) shows that character-level modules can be further enhanced by context information, leading to even better performance. To verify the power of context information, we include this context-aware model in our experiments, and further explore its potential.

Furthermore, we propose a novel word-wise dropout strategy to regularize word embeddings fine-tuning. Embedding methods can easily scale to the massive amount of text, and help the sequence labeling model better handle out-of-training-vocabulary (OOTV) words. Most of existing methods

[1] The code is available at `https://github.com/`

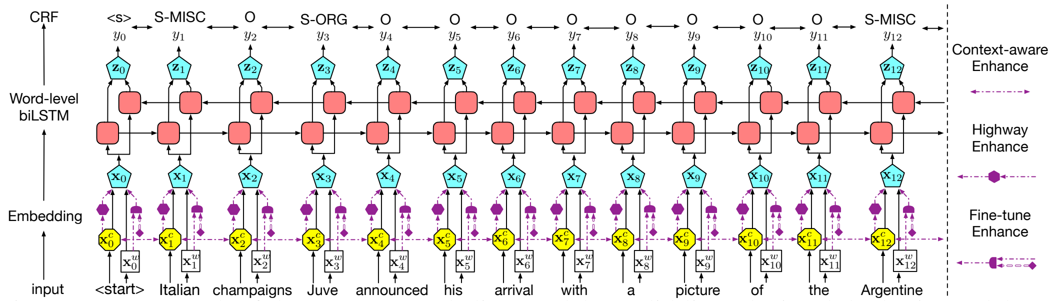
CRF
<s>
S-MISC
O
S-ORG
O
O
O
O
O
O
O
O
S-MISC
y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12}
$\mathbf{z}_0$ … $\mathbf{z}_{12}$
Word-level biLSTM
$\mathbf{x}_0$ … $\mathbf{x}_{12}$
Embedding
$\mathbf{x}^c_0$ … $\mathbf{x}^c_{12}$
$\mathbf{x}^w_0$ … $\mathbf{x}^w_{12}$
input
<start>
Italian
champaigns
Juve
announced
his
arrival
with
a
picture
of
the
Argentine
Context-aware Enhance
Highway Enhance
Fine-tune Enhance

Figure 1: Framework of the Sequence Labeling models. Solid lines depicted the basic model (B-LSTM-CRF) and dashed lines visualized several enhancements.

directly fine-tune these embeddings, and conduct training on a shifted semantic space (Collobert et al., 2011; Ma & Hovy, 2016; Lample et al., 2016). However, due to the limited size of the annotated corpus and the heavy-tailed word distribution, only a small portion of word embeddings would be updated while most of the embeddings would stay the same. Consequently, un-updated (OOTV) pre-trained embeddings might be incompatible with these neural models. To better manipulate pre-trained word embeddings, we tried to fine-tune pre-trained embeddings, but also keep informing the model about original embeddings.

Our major contributions are: (1) empirically evaluating popular state-of-the-art sequence labeling frameworks by extensive controlled experiments; (2) proposing a novel word-wise dropout strategy to better fine-tune pre-trained word embeddings; and (3) achieving the new state-of-the-art on two benchmark datasets, obtaining 91.82±0.23 F_1 for NER and 97.56±0.03 for POS-tagging without any multi-task training or external resource.

The rest of the paper is organized as follows. Section 2 briefly reviews related literature. In Section 3, we present the sequence labeling architecture used in our experiments. Character-level representation components would be compared and discussed in Section 4, and strategies for fine-tuning word embedding would be introduced in Section 5. In the end, the paper is concluded in Section 6.

2 Related Work

Most neural sequence labeling models follow the architecture in Fig. 1 and propose different modules to manipulate representations. Although character-level representations can capture lexical features and handle OOTV words and misspelled words, using them alone may not be sufficient. E.g., in Fig. 1, only by knowing the semantic information of the whole word "Juve", we can identify it as an organization instead of a person. Therefore, both word-level and character-level representations are crucial modules in the sequence labeling framework.

Based on the distributional hypothesis (Rubenstein & Goodenough, 1965), embedding techniques could learn distributed representations for words while retaining the semantic relations among them (Mikolov et al., 2013; Pennington et al., 2014). These methods have demonstrated good properties on capturing semantics for an extensive word dictionary, and are leveraged in the following models. For character-level representations, Kim et al. (2016) utilized CNNs and highway networks (Srivastava et al., 2015) to process character embeddings for the neural language model, which can make better predictions on OOTV words and misspelled words. Similarly, Chiu & Nichols (2016) employed CNN to provide character-level representations, but still relied on lexicon and did not incorporate CRF. Later, Ma & Hovy (2016) also utilized CNN for character-level representations and further added a CRF layer to handle the label dependency. In addition to CNN, LSTM has also been utilized for character-level representations. Lample et al. (2016) adopted LSTM networks for character-level representations and adopted the CRF layer used in the BI-LSTM-CRF (Huang et al., 2015). Also, recent studies leveraged neural language models to obtain better word and/or character representations, and thus lead to better performance (Liu et al., 2017; Rei, 2017; Peters et al., 2017).

Learning Rate	Momentum Factor	Dropout Ratio	Character Embedding #	Word-level State #	Word-level LSTM layer #
$\frac{0.01}{1+0.05\cdot t}$	0.9	0.5	30	300	1

Table 1: Hyper-parameters of the BI-LSTM-CRF Framework (# refers to layer numbers for LSTM, and dimension size for embedding or state)

Besides, there are other structures and strategies proposed for sequence labeling. Strubell et al. (2017) employed iterated dilated CNN instead of LSTM to handle sequence input, however it only captures word-level information and fails to beat most of previous models.

3 Neural Sequence Labeling Architecture

Before diving into the comparison of existing frameworks or further improvements, we'd like to first introduce the architecture of our implementations. Our implementations are based on the PyTorch library[2]; and each experiment has been conducted 10 times on GeForce GTX 1080 GPUs.

As discussed in Sec. 2, most neural sequence labeling models follow the framework visualized in Fig. 1. Accordingly, we utilize the BI-LSTM-CRF model for encoding, and focus on exploring different representation learning modules. For a sentence with annotations $\mathbf{y} = (y_0, \ldots, y_n)$, its word-level input is marked as $\mathbf{v} = (v_0, v_2, \ldots, v_n)$, where v_i is the i-th word; its character-level input is recorded as $\mathbf{c} = (c_{0,\_}, c_{1,1}, c_{1,2}, \ldots, c_{1,\_}, c_{2,1}, \ldots, c_{n,\_})$, where $c_{i,j}$ is the j-th character for word v_i and $c_{i,\_}$ is the space character after v_i. We record the word-level and character-level representations for word v_i as $\mathbf{x}_i^w$ and $\mathbf{x}_i^c$. And $\mathbf{X} = (\mathbf{x}_0, \ldots, \mathbf{x}_n)$ and $\mathbf{Z} = (\mathbf{z}_0, \ldots, \mathbf{z}_n)$ are the inputs and outputs of the bidirectional LSTM, where $\mathbf{x}_i = (\mathbf{x}_i^w, \mathbf{x}_i^c)$ is the representation of word v_i.

Since character-level representations are utilized to extract lexical features, we keep the character-level input case sensitive and convert word-level input to lowercase. We use GloVe 100-dimensional embeddings (Pennington et al., 2014) as pre-trained word embeddings, because it is lowercase and Ma & Hovy (2016) reported it to be the most effective one among several choices. Also, we replace rare words (i.e., frequency less than 5) with a special token (<UNK>).

The CRF describes the probability of generating the whole label sequence with regard to $\mathbf{Z}$. That is,

$$p(\hat{\mathbf{y}}|\mathbf{Z}) = \frac{\prod_{j=1}^{n} \phi(\hat{y}_{j-1}, \hat{y}_j, \mathbf{z}_j)}{\sum_{\mathbf{y}' \in \mathbf{Y}(\mathbf{Z})} \prod_{j=1}^{n} \phi(y'_{j-1}, y'_j, \mathbf{z}_j)} \tag{1}$$

where $\phi()$ is the potential function, $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_n)$ is a generic label sequence and $\mathbf{Y}(\mathbf{Z})$ is the set of all generic label sequences. Ma & Hovy (2016) defined $\phi()$ as $\phi_l(y_{j-1}, y_j, \mathbf{z}_j) = \exp(W_{y_{j-1},y_j}\mathbf{z}_i + b_{y_{j-1},y_j})$ and Lample et al. (2016) defined it as $\phi_s(y_{j-1}, y_j, \mathbf{z}_j) = \exp(W\mathbf{z}_i + b_{y_{j-1},y_j})$, where W_{y_{j-1},y_j} and b_{y_{j-1},y_j} are the weight and bias parameters corresponding to the label pair (y_{j-1}, y_j), and W is the weight parameter used for all cases. We refer the CRF module with $\phi_l()$ and $\phi_s()$ as CRF$_L$ and CRF$_S$ respectively, and we can notice that $\phi_l()$ contains more parameters and has more capability than $\phi_s()$.

To conduct training, we minimize the negative log-likelihood as the object function,

$$\mathcal{J}_{CRF} = -\sum_i \log p(\mathbf{y}_i|\mathbf{Z}_i) \tag{2}$$

For annotating, we maximizes the likelihood to find the optimal sequence $\mathbf{y}^*$,

$$\mathbf{y}^* = \arg\max\nolimits_{\mathbf{y}\in\mathbf{Y}(\mathbf{Z})} p(\mathbf{y}|\mathbf{Z}) \tag{3}$$

Although Eq. 1, Eq.2, and Eq.3 are complicated, the Viterbi algorithm can calculate them efficiently.

As to the training of neural networks, dropout is used in every layer, and stochastic gradient descent with momentum is adopted for optimization. Based on previous studies (Ma & Hovy, 2016; Liu et al., 2017), we empirically set hyper-parameters as Table 1.

For evaluation, we conduct experiments on two benchmark datasets:

[2] `http://pytorch.org/`

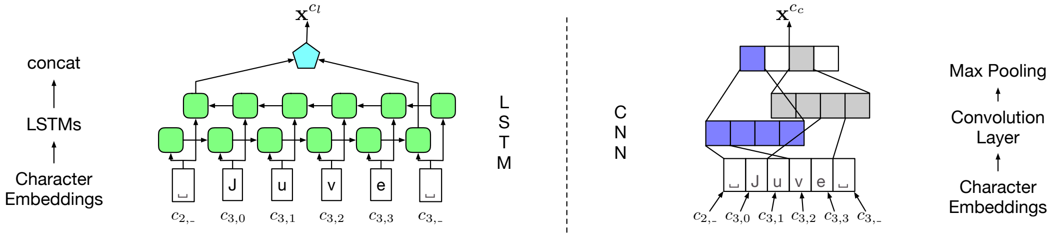

Figure 2: Character-level representation modules. The left one is LSTM (Lample et al., 2016) and the right one is CNN (Ma & Hovy, 2016)

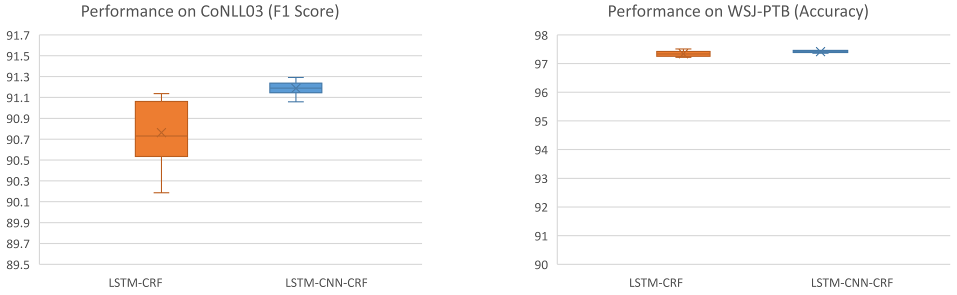

Figure 3: Performance of LSTM-CRF and LSTM-CNN-CRF with default parameters

- **WSJ-PTB**, i.e., the Wall Street Journal portion of Penn Treebank POS-tagging dataset (Marcus et al., 1993), categorizes each word into one of the 45 POS tags. The dataset has 25 sections and we use sections 0-18 as training data, sections 19-21 as development data, and sections 22-24 as test data (Ma & Hovy, 2016; Manning, 2011).
- **CoNLL03 NER** defines four entity types: `PER` (Person), `LOC` (Location), `ORG` (Organization), and `MISC` (Miscellaneous) (Tjong Kim Sang & De Meulder, 2003). In this dataset, we adopted its original training, development, and test separation.

Following the previous work (Ratinov & Roth, 2009), we adopted the BIOES schema for CoNLL03 datasets, use entity-level F_1 score for CoNLL03 dataset and Accuracy for WSJ dataset.

4 CHARACTER-LEVEL REPRESENTATION

In this section, we will first compare existing character-level representation modules using controlled experiments, then further explore its structures to improve the performance.

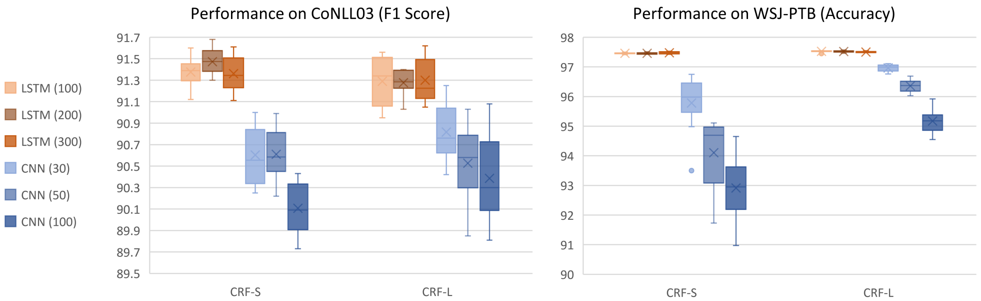

Figure 4: Performance of B-L-LSTM-CRF and B-C-LSTM-CRF. The adopted CRF is indicated by X-axis. The hidden size of character-level LSTM in B-L-LSTM-CRF is referred as "LSTM (#)" and the filter number of CNN in B-C-LSTM-CRF is referred as "CNN (#)".

Parameters		Performance (mean ± std) on Different Datasets			
LSTM (#)	CRF	CoNLL03 (F_1 score)		WSJ-PTB (Accuracy)	
		H-L	B-L	H-L	B-L
100	CRF$_L$	**91.36±0.16**	91.29±0.23	**97.53±0.03**	97.52±0.03
200	CRF$_L$	**91.36±0.08**	91.28±0.11	**97.53±0.03**	97.52±0.03
300	CRF$_L$	**91.36±0.17**	91.30±0.20	**97.52±0.02**	97.50±0.02
100	CRF$_S$	91.30±0.22	**91.38±0.14**	**97.48±0.02**	97.46±0.02
200	CRF$_S$	91.35±0.28	**91.47±0.12**	**97.48±0.02**	97.46±0.03
300	CRF$_S$	91.23±0.15	**91.36±0.17**	**97.49±0.04**	97.48±0.04

Table 2: Performance Comparison between H-L-LSTM-CRF (referred as "H-L") and B-L-LSTM-CRF (referred as "B-L"). Highlight refers to the winning setting between "H-L" and "B-L" . The hidden dimension of character-level LSTM is marked as LSTM (#).

4.1 Comparison among Existing Character-Level Representation Modules

As discussed before, we re-implemented LSTM-CRF and LSTM-CNN-CRF with our pipeline, recorded as B-L-LSTM-CRF and B-C-LSTM-CRF. They use one layer of LSTM or CNN (as visualized in Fig. 2) to obtain character-level representations, $\mathbf{x}_i^c$, then calculate $\mathbf{x}_i$ following the "basic" module depicted in Fig. 6. The performance statistics of B-C-LSTM-CRF and B-L-LSTM-CRF are summarized in Fig. 4 (for unmentioned hyper-parameters, we adopted their default values). We also conduct experiments with original model implementations and summarize their performance in Fig. 3 for comparison.

The results are quite interesting. Across different implementations, B-L-LSTM-CRF achieved the best performance. However, B-C-LSTM-CRF achieved almost the worst, and its only difference compared to B-L-LSTM-CRF is its character-level representation module. At the same time, LSTM-CNN-CRF achieves better performance than LSTM-CRF In their original implementations. Also, we can observe that, in most cases, CRF$_L$ (originally adopted by LSTM-CNN-CRF) has a marginal improvement over CRF$_S$ (originally adopted by LSTM-CRF). Accordingly, we think the reported improvement of LSTC-CNN-CRF over LSTM-CRF mainly comes from its infrastructures and the slightly better CRF module instead of the different character-level representation module.

Besides, highway network (Srivastava et al., 2015) has demonstrated its effectiveness in combining character-level with word-level (Liu et al., 2017; Kim et al., 2016). So, we append one layer of highway network to B-L-LSTM-CRF and B-C-LSTM-CRF, and marked the resulting variants as H-L-LSTM-CRF and H-C-LSTM-CRF. Their performances are summarized in Table 2 and Table 3. We can observe that the highway network could improve the performance of CNN-based modules across different settings and tasks. But for LSTM-based modules, it can only marginally improve the performance on the WSJ-PTB dataset. And for CoNLL03 dataset, it does not change the performance significantly.

In summary, we observed that LSTM is more effective than CNN serving as the character-level representation module. Also CRF$_L$ and highway networks could brought some improvements to certain degree. Therefore, we will stick to the combination of LSTM, CRF$_L$, and highway networks in the following experiments.

4.2 Exploring the Potential of LSTM

Now, let's move to further explore the potential of LSTM-based character-level representation. Since current implementations have achieved very high performance on the WSJ-PTB dataset, we will focus on the CoNLL03 dataset, and evaluate our final model on both datasets in the end.

Previous frameworks, such as LSTM-CRF and LSTM-CNN-CRF leverage character-level representations in an embedding manner, which is context-agnostic. However, context information is crucial for lexical features, e.g., capitalization has different meaning for the first word and other words. Liu et al. (2017) proposed to manipulate character-level representations in an character-level lan-

Parameters		Performance (mean ± std) on Different Datasets			
CNN (#)	CRF	CoNLL03 (F_1 score)		WSJ-PTB (Accuracy)	
		H-C	B-C	H-C	B-C
30	CRF$_L$	**90.98±0.24**	90.82±0.26	**97.09±0.30**	96.96±0.11
50	CRF$_L$	**91.04±0.23**	90.53±0.35	**97.14±0.14**	96.36±0.21
100	CRF$_L$	**90.94±0.19**	90.39±0.42	**97.01±0.28**	95.18±0.40
30	CRF$_S$	**90.97±0.43**	90.60±0.26	**96.27±0.34**	95.78±0.95
50	CRF$_S$	**90.91±0.07**	90.61±0.24	**96.38±0.43**	94.10±1.16
100	CRF$_S$	**90.62±0.36**	90.11±0.23	**96.00±0.61**	92.01±3.09

Table 3: Performance Comparison between H-C-LSTM-CRF (referred as "H-C") and B-C-LSTM-CRF (referred as "B-C"). Highlight refers to the winning setting between "H-C" and "B-C". The filter number of CNN is marked as CNN (#).

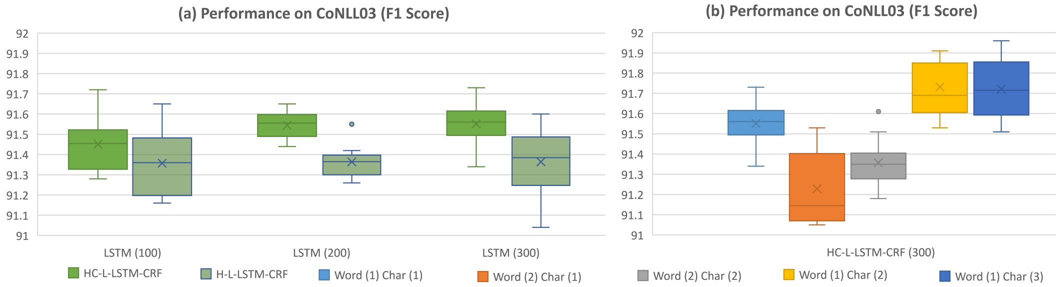

Figure 5: Performance of HC-L-LSTM-CRF and H-L-LSTM-CRF. The hidden size of character-level LSTM is referred as "LSTM (#)". The layer numbers of word-level and character-level LSTM are referred as "Word(#) Char(#)". (a) compares HC-L-LSTM-CRF with H-L-LSTM-CRF and (b) compares different settings of HC-L-LSTM-CRF.

guage model manner. As visualized in Fig. 1, it would treat characters of the whole sentence as a sequence instead of a single word. We refer the context-aware LSTM-based model with highway enhancement as HC-L-LSTM-CRF, and its performance is summarized in Fig. 5 (a). We can observe that HC-L-LSTM-CRF outperformed H-L-LSTM-CRF in all three settings and achieved the best performance when the state size of character-level LSTM is set to 300.

Moreover, since many tasks have greatly benefited from deeper models (He et al., 2016), we try to increase the depth of LSTM layers for better performance. The performance of several depth combinations is summarized in Fig. 5 (b). We can find that increasing the depth of word-level biLSTM could be harmful to the performance, while increasing the depth of character-level LSTMs could boost the average F_1 score over 91.7. We think this phenomenon implies that the word-level structures are more vulnerable than character-level LSTMs, and are also more easier to overfit.

5 Word-level Representation

So far, we have explored existing character-level representation components. As summarized in Fig. 6, the "Basic" module refers to B-LSTM-CRF, the "Char" module refers to HC-LSTM-CRF. Other module tries to further improve the HC-LSTM-CRF model by regularizing the fine-tuning of word embeddings. In this section, we would fix the hyper-parameters of HC-LSTM-CRF to the value achieving the best performance in the last section[3], which is referred as "Vanilla" in this section.

Embedding methods can be applied to a massive amount of text and provide distributional representations for an extensive dictionary of words. By leveraging such embeddings, sequence labeling models could have more potential to handle rare words and OOTV words. For example, in Fig. 1, the word "Juve" does not show up in the training corpus of the CoNLL03 dataset but it is included in the

[3] two layers of character-level LSTM, whose state size is 300

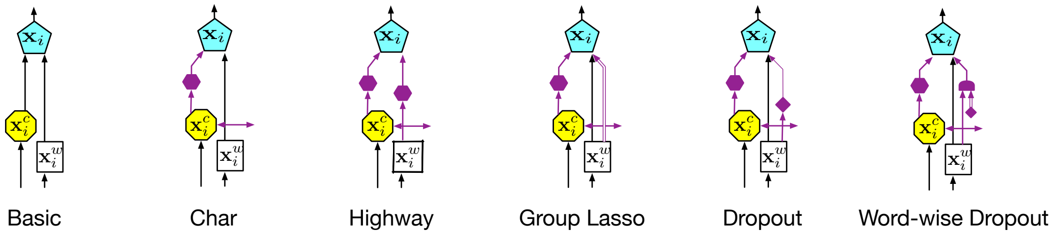

Figure 6: Word Representation Modules

dictionary of pre-trained embeddings. Leveraging such embedding, our NER model can correctly annotate it as an ORG (organization) entity instead of PER (person) entity. Consequently, the choice of pre-trained embeddings has a big impact on the resulting performance (Ma & Hovy, 2016).

Most existing frameworks use these embeddings as initialization, and crudely update their in-corpus portion during training. Accordingly, the resulting sequence labeling model would be trained on a shifted semantic space instead of the original one. In this section, we explore several strategies to reduce the gap between the shifted and original semantic spaces.

5.1 Frozen Word Embeddings

In order to reduce the gap caused by fine-tuning in-corpus words' embeddings, a possible strategy is to shift the whole semantic space instead of single vectors. That is, instead of updating word embeddings during training, we directly learn a mapping from the original semantic space to the modified semantic space, and not update pre-trained embeddings during training. Being aware of the effectiveness of highway layers in character-level representations, we considered to append a highway layer upon the word embedding layer to conduct this transformation (as depicted as "Frozen" in Fig. 6). We refer the model with frozen word embeddings but no highway layer as "Frozen", and the model with both as "FH".

We apply previous strategies on the Vanilla model, and summarize their performance in Table 4. From the results, we find both "Frozen" and "FH" could be harmful to the performance, and appending the highway network to word embeddings could make the performance even worse. We think this phenomenon also implies the fragile of word-level structures. Also, the performance demonstrates that the fine-tuning of word embedding is a highly non-linear transformation, and thus is hard to be directly leaned as a function.

5.2 Update Part of Embeddings

Alternatively, we've also considered to fine-tune only part of in-corpus pre-trained embeddings, while leaving others unchanged. Ideally, we'd like to only fine-tune common words' embeddings while leaving entity names' embeddings unchanged. Intuitively, this strategy could allow the model fine-tune the embeddings but also be friendly to those OOTV words.

To achieve this goal, we rewrite word representation for word v_i as $\mathbf{x}_i^w = \mathbf{x}_i^{w*} + \mathbf{x}_i^{w\prime}$, where $\mathbf{x}_i^{w*}$ is the pre-trained embedding and $\mathbf{x}_i^{w\prime}$ is the fine-tuning shift. Then, not fine-tuning all of in-corpus embedding can be formulated as group sparsity on $\mathbf{x}_i^{w\prime}$. Accordingly, we add group lasso (Meier et al., 2008) to the objective function, and the resulting objective becomes

$$\min \mathcal{J}_{CRF} + \lambda \cdot \sum_{v_i} |\mathbf{x}_i^{w\prime}|_2$$

In order to train the neural network with this objective, we adopt the stochastic proximal gradient descent, where the proximal operator is

$$\text{prox}_{\lambda,\alpha_t} = \frac{\mathbf{x}_i^{w\prime}}{|\mathbf{x}_i^{w\prime}|_2} \max\{0, |\mathbf{x}_i^{w\prime}|_2 - \lambda\alpha_t\}$$

In table 4, we denote this strategy as "GLasso", and report its performance under two different λ values. When setting λ to 1×10^{-4} and 3×10^{-4}, the averaged number of fine-tuned embeddings

Statistics	Vanilla	Frozen	FH	GLasso (λ)		Ele-wise (ratio)		Word-wise (ratio)	
				1×10^{-4}	3×10^{-4}	0.2	0.3	0.2	0.3
mean	91.68	91.49	91.09	91.77	91.72	91.69	91.72	91.82	91.82
std	0.13	0.14	0.19	0.13	0.19	0.15	0.19	0.23	0.17
max	91.91	91.78	91.31	91.94	91.89	91.88	92.03	92.19	92.07

Table 4: Performance of Different Fine-tuning Strategy on CoNLL03

	IV	OOTV	OOEV	OOBV
Ratio of Entity Word	14.03%	51.89%	0.05%	24.85%
Number of Entity Word	5795	2077	6	284
F_1 score of Vanilla	91.15	92.12	100	78.42
F_1 score of Word-wise (ratio set to 0.2)	91.25	92.15	100	78.56
F_1 score of Word-wise (ratio set to 0.3)	91.24	92.21	100	78.41

Table 5: Performance of Word-wise Dropout on CoNLL03

are 8,625.3 and 7,649.6. Also, we found the CoNLL03 training corpus has about 21,009 different words (converted to lowercase), and setting λ to these two values would allow the model to fine-tune most of frequent words. And by increasing the value of λ, we found the model would fine-tune less embeddings, but the performance would drop to a lower value.

Despite the additional hyper-parameter λ, this strategy can only marginally improve the performance, which is not very effective. We argue that this is because frequent words composed most of the corpus, and fine-tuning their embeddings would still result in a shifted semantic space.

5.3 Word-wise Dropout

In the end, we try to let the model fine-tune all embeddings, but also let the model being aware of the original pre-trained embeddings during fine-tuning. Specifically, during training, we will randomly recover part of fine-tuned embeddings to their corresponding pre-trained embeddings. In other words, we would randomly set $\mathbf{x}_i^{w\prime}$ to zero for some i. This strategy can be interpreted as applying an additional "Word-wise" dropout on $\mathbf{x}_i^{w\prime}$.

We summarize its performance with different dropout ratios in Fig. 7 and Table 4. We can find its averaged F_1 score exceeds 91.8 when the dropout ratio is set to 0.2 or 0.3. Besides, we further conduct experiments to evaluate its performance on four kinds of words, i.e., In-Both-Vocabulary (IV), Out-of-Training-Vocabulary (OOTV), Out-of-Embedding-Vocabulary (OOEV) and Out-of-Both-Vocabulary (OOBV). Table 5 informs the statistics and performance of the partition on CoNLL03. We can observe the word-wise dropout helps the sequence labeling model generalize better on the OOTV partition Also, we can observe that this strategy helps the model reduce the overfitting in the IV partition.

Additionally, we try to apply the normal dropout on $\mathbf{x}_i^{w\prime}$. It would randomly recover $\mathbf{x}_i^{w}$ to $\mathbf{x}_i^{w*}$ in a dimension-wise manner instead of a word-wise manner. We refer the corresponding model of this strategy as "Ele-wise", and listed its performance in Table 4. We can observe that it could improve the performance, but is less effective comparing to the "Word-wise" strategy or the "GLasso" strategy.

5.4 Performance of Final Models

We've discussed several strategies to fine-tune pre-trained word embeddings, and the most effective strategy on the CoNLL03 dataset is the word-wise dropout strategy with the dropout ratio 0.2. We denote this model as WC-LSTM-CRF.

Its performance has been summarized in Table 6. We can observe that it outperformed existing methods and achieved the new state-of-the-art.

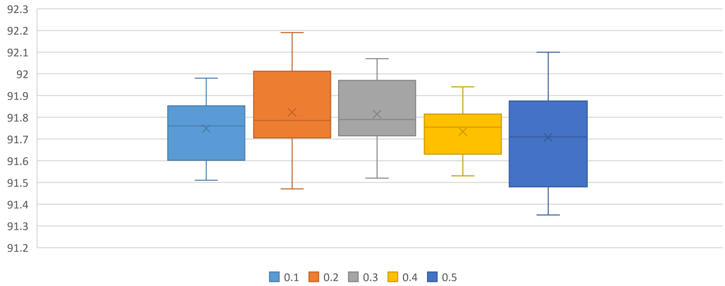

Figure 7: Performance of Word-wise Dropout on CoNLL03 with different Dropout Ratios

Models	Performance on Different Datasets			
	CoNLL03 (F_1 score)		WSJ-PTB (Accuracy)	
	mean ± std	max	mean ± std	max
WC-LSTM-CRF	91.82±0.23	92.19	97.56±0.03	97.64
HC-L-LSTM-CRF	91.68±0.13	91.91	97.50±0.02	97.52
B-L-LSTM-CRF	91.47±0.12	91.68	97.52±0.03	97.55
B-C-CNN-CRF	90.82±0.26	91.25	96.96±0.23	97.11

Table 6: Performance Comparison between H-C-LSTM-CRF (referred as "H-C") and B-C-LSTM-CRF (referred as "B-C"). Highlight refers to the winning setting between "H-C" and "B-C". The filter number of CNN is marked as CNN (#).

6 Conclusion

As a conclusion, we first conduct thorough examinations of representation modules in existing models. From the results, we observe LSTM-based model is more effective than CNN-based models. Furthermore, we explore the potential of character-level representations and propose several strategies to regularize the fine-tuning for pre-trained embeddings. Eventually, our resulting model achieves the new state-of-the-art on two benchmark datasets. In the future, we plan to utilize the word-wise dropout to help fine-tune embeddings for more complicated tasks like relation extraction.

References

Jason P. C. Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *TACL*, 2016.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, 2011.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *CoRR*, 2015.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pp. 2741–2749, 2016.

John D. Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

Guillaume Lample, Miguel Ballesteros, Kazuya Kawakami, Sandeep Subramanian, and Chris Dyer. Neural architectures for named entity recognition. In *NAACL-HLT*, 2016.

L. Liu, J. Shang, F. Xu, X. Ren, H. Gui, J. Peng, and J. Han. Empower Sequence Labeling with Task-Aware Neural Language Model. *arXiv:1709.04109*, 2017.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *ACL*, 2016.

Christopher D Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, 2011.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 1993.

Lukas Meier, Sara Van De Geer, and Peter Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. *arXiv:1705.00108*, 2017.

Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *CoNLL*, 2009.

Marek Rei. Semi-supervised multitask learning for sequence labeling. In *ACL*, 2017.

Nils Reimers and Iryna Gurevych. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *arXiv preprint arXiv:1707.06799*, 2017a.

Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. *arXiv preprint arXiv:1707.09861*, 2017b.

Herbert Rubenstein and John B Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *NAACL-HLT*, 2003.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. Fast and accurate sequence labeling with iterated dilated convolutions. *arXiv preprint arXiv:1702.02098*, 2017.

Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Natural language learning at NAACL-HLT*, 2003.

APPENDIX

F_1 score of Experiments on CPU			F_1 score of Experiments on GPU		
Mean	std	Max	Mean	std	Max
90.78	0.25	91.16	91.19	0.07	91.29

Table 7: Performance Statistics of Original LSTM-CNN-CRF Implementation on CoNLL03 NER.

In their original papers, LM-LSTM-CRF is reported to perform better than LSTM-CRF, however, Reimers & Gurevych (2017b) failed to reproduce this results even with the code released by the paper authors.

To figure out the reason of this discordance, we checked out the implementation[4] released by the author of LSTM-CNN-CRF. After some experiments, we found the discordance may due to the subtle difference of runtime experiments. We summarize the performance statistics on two different environments, CPU and GPU, in Table 7. When conducting training on GPU, the averaged F_1 score reaches 91.19, which is similar to the score reported in the original paper, 91.21. However, after swifting to CPU, the performance drops to 90.79, which is similar to the score reported in Reimers & Gurevych (2017b)[5]. This phenomenon implies the huge effect of infrastructures and runtime environments, and makes it even more necessary to conduct comparison with controlled experiments.

[4] `https://github.com/XuezheMax/LasagneNLP`

[5] the author of Reimers & Gurevych (2017b) confirmed their experiments about LSTM-CNN-CRF were conducted on CPU.