Raw-to-End Name Entity Recognition in Social Media

Liyuan Liu[†] **Zihan Wang**[†] **Jingbo Shang**[†] **Dandong Yin**[†] **Heng Ji**[†]
Xiang Ren[♯] **Shaowen Wang**[†] **Jiawei Han**[†]
[†] University of Illinois at Urbana-Champaign, Urbana, IL, USA
[♯] University of Southern California, Los Angeles, CA, USA
[†]{ll2, zihanw2, shang7, dyin4, hengji, shaowen hanj}@illinois.edu [♯]xiangren@usc.edu

Abstract

Taking word sequences as the input, typical named entity recognition (NER) models neglect errors from pre-processing (*e.g.*, tokenization). However, these errors can influence the model performance greatly, especially for noisy texts like tweets. Here, we introduce Neural-Char-CRF, a raw-to-end framework that is more robust to pre-processing errors. It takes raw character sequences as inputs and makes end-to-end predictions. Word embedding and contextualized representation models are further tailored to capture textual signals for each character instead of each word. Our model neither requires the conversion from character sequences to word sequences, nor assumes tokenizer can correctly detect all word boundaries. Moreover, we observe our model performance remains unchanged after replacing tokenization with string matching, which demonstrates its potential to be tokenization-free. Extensive experimental results on two public datasets demonstrate the superiority of our proposed method over the state of the art.[1]

1 Introduction

The explosively growing social media generates noisy, irregular texts on a massive scale. How to digest such fast-evolving text data has become an urgent, challenging topic and attracted a significant amount of attention (Strauss et al., 2016; Derczynski et al., 2017). Here we focus on one important step towards extracting information from social media posts: named entity recognition (NER).

The key challenge of NER in social media lies in the noisy nature of online posts. Compared to regular articles (*e.g.*, the Wall Street Journal news), these posts are written in a more informal and creative manner. Hence, even for pre-processing (*e.g.*, tokenization), it becomes difficult to build tools to analyze such texts with no or little errors. Moreover, pre-processing is crucial to get good NER performance, and thus their errors cannot be neglected. For example, on the Broad Twitter Corpus (Derczynski et al., 2016), we observe word boundaries of more than 45% named entities to be incorrectly identified by spaCy (Honnibal and Montani, 2017). At the same time, as in Figure 1, the pipeline system builds pre-processing and sequence labeling separately, assumes all entity boundaries are correctly identified as word boundaries, and are vulnerable to pre-processing errors. For example, when the tokenizer wrongly identifies "Lakers-Raptors" as one token, the NER model has to make one prediction for the whole string and cannot correctly recognize "Lakers" and "Raptors" as two entities.

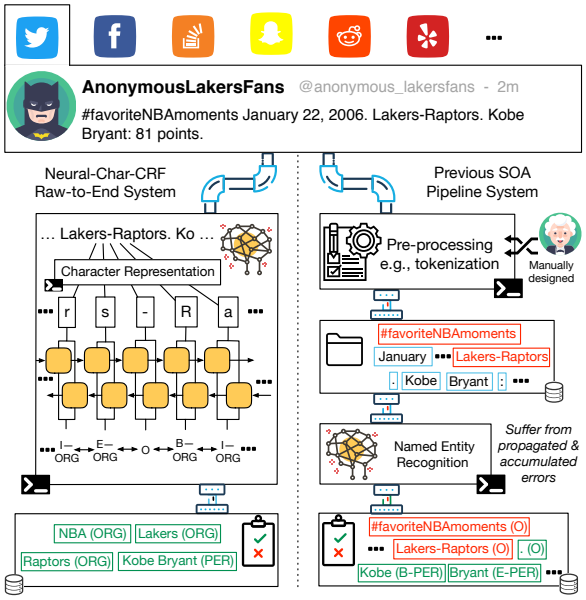


Figure 1: Comparison between Pipeline System and Neural-Char-CRF. In the pipeline system, pre-processing errors hurt the model performance. The raw-to-end training in Neural-Char-CRF suffers less from such error propagation and reduces human endeavors.

[1]The implementations and datasets are made available at: https://github.com/LiyuanLucasLiu/Raw-to-End.

In the past few years, many efforts have been made to unify different stages in the pipeline system and these approaches have demonstrated a great potential to reduce human endeavors, alleviate error propagation, and improve the performance. For example, neural networks achieve good success on jointly extracting textual signals and detecting entities (Lample et al., 2016; Ma and Hovy, 2016). Still, these methods take word sequences as inputs and neglect errors from pre-processing. For example, on the BTC dataset, after replacing perfect tokenizer with system tokenizer, the performance drops 24.40 absolute F_1 on average.

In this paper, we propose Neural-Char-CRF, a raw-to-end framework that takes raw character sequences as inputs and makes predictions in an end-to-end manner. It integrates pre-processing with representation learning modules: modules are designed to align each character to a pre-trained word embedding. Specifically, we introduce two strategies for building this alignment, one leverages tokenizer and the other utilizes string matching. Besides, we pre-train character-level language models and construct contextualized character representations. Different from the pipeline system, our model captures textual signals for each character instead of each word, predict whether it belongs to an entity, and if so what its position and type are. For example, in Figure 2, our model needs to construct representations and make predictions for each character in "Lakers-Raptors". Therefore, if the tokenizer wrongly detect the whole string as one token, our model would need to handle noisy representations, but it still has the potential to correctly recognize "Lakers" and "Raptors" as entities.

We evaluate our model on two public datasets — Twitter Name Tagging (TNT) (Lu et al., 2018) and Broad Twitter Corpus (BTC) (Derczynski et al., 2016). Experimental results show the effectiveness of our proposed method, which advances state-of-the-art by 3.70 and 6.65 absolute F_1 gain on TNT and BTC respectively. We also observe that, the performance of our model remains unchanged after replacing tokenization with string matching, which demonstrates the potential of our model to be tokenization-free.

2 Neural-Char-CRF

In this section, we first introduce the raw-to-end problem formulation, then proceed to the proposed Neural-Char-CRF framework.

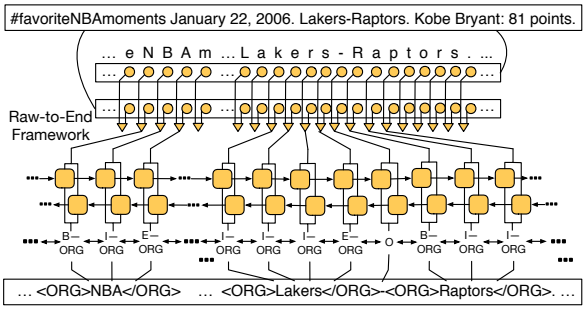


Figure 2: The proposed Neural-Char-CRF Framework. It accepts the raw text as the input and makes predictions at the character level.

2.1 Problem Formulation

We formulate the raw-to-end NER as a character-level sequence labeling task. The input is a character sequence $X = \{x_1, x_2, \ldots, x_T\}$, where x_i $(1 \leq i \leq T)$ is the i-th character and T is the input length. Following the `IOBES` labeling schemes (Ratinov and Roth, 2009), we assign a label for each character. Specifically, when a sequence of characters is identified as a named entity, its starting, middle and ending character are labeled as `B-`, `I-`, and `E-` respectively (if the entity has only one character, it is labeled as `S-`), followed by the type; otherwise, it would be labeled as `O` instead. Referring the label for x_i as y_i, the goal of the raw-to-end NER is to predict the label sequence $Y = \{y_1, y_2, \ldots y_T\}$.

2.2 Framework Architecture

We propose a novel raw-to-end framework, Neural-Char-CRF, to reduce the error propagation and the reliance on pre-processing. This framework directly takes character sequences as inputs and makes prediction for each character. As visualized in Figure 2, Neural-Char-CRF first constructs representations for each character, then detects entities with LSTM-CRF.

Character Representation. Recent state-of-the-art NER models usually unify representations learned by multiple methods. We assume there are n different representation modules, namely M_i $1 \leq 1 \leq n$ (*e.g.*, different pre-trained word embedding or contextualized representation models). Given the j-th character in the input sequence, the representation vector produced by module M_i is denoted as $\mathbf{f}_{i,j}$. In this paper, we concatenate the output from different modules as the final representation, *i.e.*, $\mathbf{f}_j = [\mathbf{f}_{1,j}; \mathbf{f}_{2,j}; \cdots; \mathbf{f}_{n,j}]$. Given the input sequence X, we define its representation sequence as $\mathcal{F} = \{\mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_T\}$.

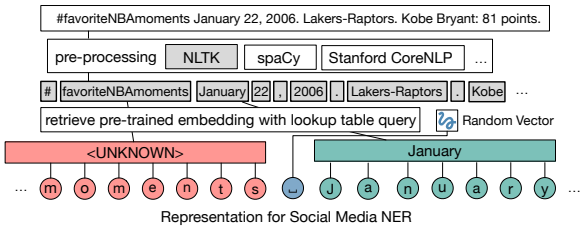


Figure 3: Pre-processing tools are leveraged to integrate pre-trained word embedding in a robust manner.

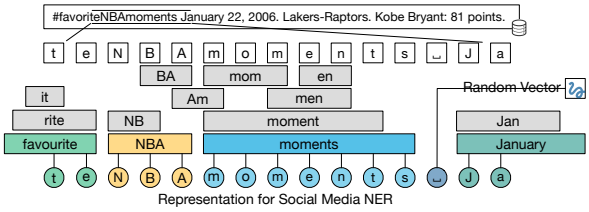


Figure 4: We used string matching to align the dictionary to the raw text for embedding learning.

Decoding with LSTM-CRF. Building upon the character representation of the input sequence, we use LSTM-CRF (Huang et al., 2015) to extract entities: we first feed $\mathcal{F}$ into Bi-LSTMs, whose output is marked as $\mathcal{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_T\}$. A linear-chain CRF is further leveraged to model the whole label sequence simultaneously. Specifically, for the input sequence $\mathcal{Z}$, CRF defines the conditional probability of $Y = \{y_1, \cdots, y_T\}$ as

$$p(Y|\mathcal{Z}) = \frac{\prod_{t=1}^{T} \phi(y_{t-1}, y_t, \mathbf{z}_t)}{\sum_{\hat{Y} \in \mathcal{Y}(\mathcal{Z})} \prod_{t=1}^{T} \phi(\hat{y}_{t-1}, \hat{y}_t, \mathbf{z}_t)} \quad (1)$$

where $\hat{Y} = \{\hat{y}_1, \cdots, \hat{y}_T\}$ is a possible label sequence, $\mathcal{Y}(\mathcal{Z})$ refers to the set of all possible label sequences for $\mathcal{Z}$, and $\phi(y_{t-1}, y_t, \mathbf{z}_t)$ is the potential function of the CRF. In this paper, we define the potential function as: $\phi(y_{t-1}, y_t, \mathbf{z}_t) = \exp(W_{y_t}\mathbf{z}_t + b_{y_{t-1},y_t})$ where W_{y_t} and b_{y_{t-1},y_t} are the weight and bias.

During the model training, we use the negative log-likelihood of Equation 1 as the loss function. In the inference stage, we find the predicted label sequence for input X by maximizing the probability in Equation 1. Although the denominator in Equation 1 contains a number of terms exponential to the length T, due to the definition of the potential function, both training and inference can be efficiently conducted using dynamic programming.

3 Character Representation Modules

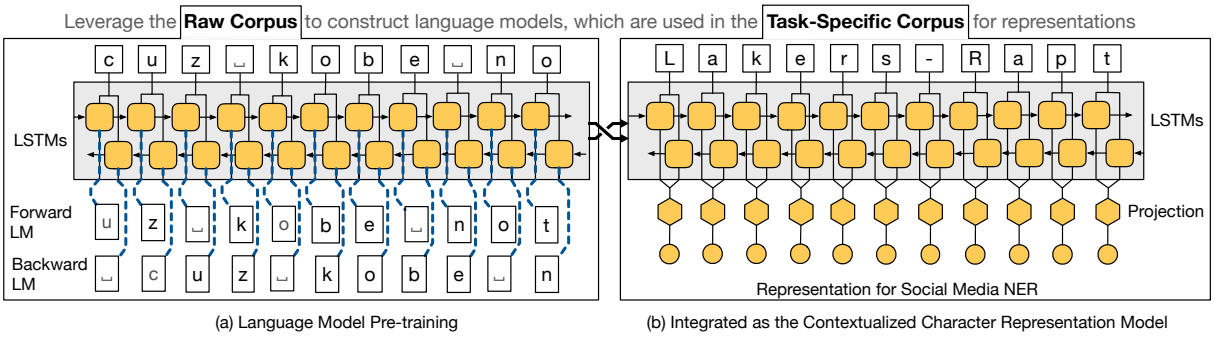
In this section, we discuss character representation modules used in Neural-Char-CRF. Representation learning techniques at the word level, such as word embedding (Mikolov et al., 2013) and contextualized representations (Peters et al., 2018; Devlin et al., 2018; Akbik et al., 2018), have demonstrated their effectiveness in NER. However, learning representation at the character level has not been explored sufficiently. We keep the same spirits of word embedding and contextualized representations, and further tailor these two techniques from the word level to the character level.

3.1 Robust Word Embedding Alignment

Existing work (Cherry and Guo, 2015) has demonstrated that word embedding trained on a web-scale corpus is incredibly powerful in improving Twitter NER performance. The key reason is that such trained embedding helps the model to better understand rare words in NER datasets. Trained on web-scale corpus, rare words will appear for a reasonably large number of times, and thus have informative embedding vectors. Since named entitie often contain rare and uncommon words, such embeddings could provide valuable information for NER. Therefore, we aim to incorporate word embedding into the character representation. We align each character to a word, which is specified by tokenization or string matching, then integrate the word embedding into the representation of this character. In this way, the NER model is aware of word-level signals while not forced to make predictions at word level. We elaborate these two approaches below.

Tokenization-based Embedding Alignment. The first approach is designed to leverage preprocessing in a robust manner. It uses tokenization results to align word embedding down to each character. The workflow is visualized in Figure 3. First, we run a tokenizer to identify the word boundaries and retrieve the pre-trained word embedding with lookup table queries. Then, for each word, we use its word embedding vector as the representation for all characters that belong to this word. At the same time, all whitespace characters use the same embedding vector, which is randomly initialized. In this strategy, the tokenizer is only used in the alignment procedure, the sequence labeling model does not depend on the detected token boundaries and is more robust to error propagation.

String Matching-based Embedding Alignment. To further reduce the reliance on pre-processing, we design a string matching-based approach to build the alignment in a tokenization-free manner.



(a) Language Model Pre-training (b) Integrated as the Contextualized Character Representation Model

Figure 5: Contextualized character representation is constructed by pre-trained language models. Specifically, as in (a), the neural models are trained by predicting the afterwards / previous words; as in (b), these models are integrated into the downstream task to provide contextualized character representations.

As shown in Figure 4, given the input character sequence, we first find all possible word matches from a given dictionary. It is very likely that some characters will be matched to multiple words. For example, the character "N" is matched to two words, "NB" and "NBA". Therefore, for alignment purpose, we need to identify the most suitable match among these words. We decide to choose the word that has the highest inverse document frequency (IDF) because words of higher IDF are more specific (Sparck Jones, 1972). Particularly, we first sort all possible matches decreasingly by their IDFs. Every time, we retrieve the match of the highest IDF for each character. In this way, we give the priority to more specific and informative words. As illustrated in Figure 4, this strategy can correctly identify the suitable match in most cases. After we finalize the matched word for each character, their word embedding vectors are used as representations for corresponding characters.

It is worth mentioning that this alignment process can be implemented in an efficient way. To be more specific, by using the union-find set, the time complexity of this process can be bounded by $O(T\alpha(T))$, where T is the input sequence length and $\alpha(\cdot)$ is the inverse Ackermann function. This function has a value $\alpha(T) < 5$ for any value of T that can be written in this physical universe, therefore the alignment process takes place in a linear time. Moreover, the string matching-based alignment has a slight advantage over the tokenization-based alignment – it requires no human endeavor to build a tokenizer.

3.2 Contextualized Char Representation

Contextualized representation learning at the word level has been widely adapted in state-of-the-art sequence labeling models. They rely on bidirectional neural language models to capture the context information before and after a certain word. This is a perfect supplementary to the context-agnostic information contained in word embedding. Therefore, we design a contextualized *character* representation model. We first train bidirectional character-level language models with a large raw corpus (in Figure 5 (a)), and then integrate them into the Neural-Char-CRF framework (in Figure 5 (b)).

We present the details of character-level language modeling and the integration as follows.

Character-Level Language Modeling. As shown in Figure 5, the bi-directional character-level language model contains two character-level language models. Character-level language modeling aims to model the probability distribution of the character sequence. Typically, the probability of the sequence $\{x_1, \cdots, x_T\}$ is defined in a "forward" manner: $p(x_1, \cdots, x_T) = \prod_{t=1}^{T} p(x_t|x_1, \cdots, x_{t-1})$. As in Figure 5, we first map the input sequence X to a list of character embedding vectors and pass them into a recurrent neural network, whose output is referred as $\mathbf{h}_t$. Then, the probability $p(x_t|x_1, \cdots, x_{t-1})$ is calculated using the softmax function. The the backward language model is the same as the forward language model, except that it decomposes the probability of the sequence $\{x_1, \cdots, x_T\}$ as $p(x_1, \cdots, x_T) = \prod_{t=1}^{T} p(x_t|x_{t+1}, \cdots, x_T)$. Its output for character x_t is denoted as $\mathbf{h}_t^r$. Both language models use the negative log-likelihood as the training objective.

Language Model Integration. Using the bidirectional character-level language models, we construct contextualized representations for each character. Due to the complexity of natural language, large dimensions of $\mathbf{h}_t$ and $\mathbf{h}_t^r$ are usually required in language models, which could be too

Table 1: Dataset Statistics of TNT and BTC.

Statistics	TNT			BTC		
	Train	Dev	Test	Train	Dev	Test
# Sent	4290	1432	1459	6261	999	1998
# Token	69K	23K	23K	98K	15K	35K
# Char	327K	108K	109K	541K	75K	190K
# PER	3053	1016	991	3031	706	1600
# LOC	978	350	307	1981	151	601
# ORG	2617	889	964	2255	270	792
# MISC	876	267	264	\	\	\

large for the NER task. To avoid over-fitting, we add a linear transformation to project $\mathbf{h}_t$ and $\mathbf{h}_t^r$ to a lower dimension. In details, we use $\mathbf{r}_t = W_{cr} \cdot [\mathbf{h}_t, \mathbf{h}_t^r] + \mathbf{b}_{cr}$, where W_{cr} and $\mathbf{b}_{cr}$ are parameters to learn during the training of NER. The output $\mathbf{r}_t$ is the contextualized representation for the character x_t, and serves as a part of its character representation.

4 Experiment

4.1 Benchmark Datasets

Here, to compare the model performance in a raw-to-end manner, we first convert word-level annotations to character-level annotations. Specifically, typical NER benchmark datasets are annotated after pre-processing, which includes but is not limited to adding/removing whitespaces by tokenization, spelling changes due to normalization, and even language changes using translation. Consequently, this process is hard to be reverted without the original texts, and we choose two benchmark datasets, as follows, with the raw texts.

- **TNT** (Lu et al., 2018) contains tweets from May 2016, January 2017 and June 2017, retrieved by using sports and social event related keywords. It is annotated with four types of entities(PER, LOC, ORG and MISC).
- **BTC** (Derczynski et al., 2016) contains tweets from various periods and regions, related to diverse topics. It is annotated with three types of entities(PER, LOC and ORG).

For both datasets, we adopt their recommended data split for training, development and testing sets, and summarize their statistics in Table 1. As to more details about the label conversion, please find the details in Appendix A, which describes the rules we applied to map the pre-processed words back to the original raw texts and export character-level annotations.

4.2 Baseline Methods

We pair state-of-the-art (Twitter) NER models with three popular pre-processing methods, which forms *twelve baselines*. Specifically, the four NER models we considered are:

1. **TwitterNER** (Mishra and Diesner, 2016) is specifically designed for tweets. It relies on not only handcrafted features, but also domain-specific resources.
2. **LSTM-CNNs-CRF** (Ma and Hovy, 2016) does not rely on feature engineering and constructs the sequence labeling in an end-to-end manner.
3. **LM-LSTM-CRF** (Liu et al., 2018b) is a sequence labeling model, which integrates language models without pre-training and uses them to construct representations.
4. **Flair** (Akbik et al., 2018) leverages language models to construct contextualized word representations and achieves the state-of-the-art performance on English NER for news corpus.

For pre-processing, we considered three tokenizers:

1. **spaCy** (Honnibal and Montani, 2017) employs the NLP package spaCy for tokenization, which features efficiency and effectiveness[2].
2. **Stanford CoreNLP** (Manning et al., 2014) leverages the widely used Stanford CoreNLP (Manning et al., 2014) as tokenizer[3].
3. **NLTK** (Bird et al., 2008) is a leading platform for building NLP pipelines and we only used its tokenization module here[4].

Since some of these pre-processing methods are not non-destructive, special handling is needed to convert entities to the original texts.

4.3 Training and Implementation Details

In our experiments, each neural model was trained with one Nvidia V100 GPU or one Nvidia 1080 Ti GPU; other models were trained on a server with 20 cores of Intel Xeon CPU E5-2680 v2 @ 2.80GHz. Our implementations would be released as a flexible framework, and more details about model training are described as below.

Language models. Following the previous work (Akbik et al., 2018), we use one-layer LSTMs with 2048 hidden states in each direction. The batch size is set to 128, and the training is conducted with backpropagation through time (BPTT) with a window size of 256. We adopted a two-

[2]The version 2.0.18 release is used.
[3]The version 3.9.2 release is used.
[4]The version 3.4 release is used.

Table 2: Performance Comparison of Sequence Labeling Models and Tokenizers on Twitter NER Benchmarks. The reported numbers are F_1 score.

Tokenizer	Methods	TNT	BTC
NLTK	TwitterNER	73.41	64.40
	LSTM-CNN-CRF	80.25	66.48
	LM-LSTM-CRF	80.85	67.73
	Flair	**83.26**	**68.33**
spaCy	TwitterNER	70.40	37.21
	LSTM-CNN-CRF	71.66	29.04
	LM-LSTM-CRF	72.25	31.17
	Flair	73.49	33.20
Stanford CoreNLP	TwitterNER	70.18	38.01
	LSTM-CNN-CRF	64.22	29.71
	LM-LSTM-CRF	63.14	28.97
	Flair	65.58	32.39
Neural-Char-CRF (Match)		**86.96**	**74.98**
Neural-Char-CRF (NLTK)		85.59	74.39

level encoding to better handle the noisy input, in which each character would be encoded as a type vector (*e.g.*, number and lower case) and a character vector, and the concatenation of these two vectors would be passed to the LSTMs. We use Adam (Kingma and Ba, 2014; Liu et al., 2019a) as the optimizer with default hyperparameters. Dropout is applied with the probability of 0.01 and the gradient is clipped at 5. During training, we would anneal the learning rate by 0.1 if the loss stops for decreasing for three epoch, and halt the training when negligible gains were observed or after one week. We collect a large raw tweet corpus for embedding learning and language model training, specifically, we collect tweets with geolocation tag in the United States from January 1 to December 31 in 2015. In total, we collect more than 700 million tweets consisting of more than 52 billion characters. With this dataset, the forward language model achieves the character-level perplexity of 3.32 on the training corpus and the backward language model achieves 3.30.

Embedding models. We build the embedding dictionary by following the previous work (Liu et al., 2015; Shang et al., 2018), a filter is further applied to remove words with the frequency less then 30. The embedding dimension is set to 100, the window size is set to 6 and the number of negative samples is set to 25. We integrate the collected raw tweets corpus and the wiki dump[5] for the model training, and optimization is conducted with SGD and stopped after 10 epochs. The resulting embedding model contains about 1 million tokens and would be used in both our method and baseline methods for a fair comparison.

[5]https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

Table 3: Different word embeddings use different tokenizers. Even for the same word embedding, different words could be tokenized differently (*e.g.*, "There's" and "That's" as below). Intuitively, inappropriate tokenization results result in sub-optimal performance.

Method \ Raw Text	`There's ...`	`That's ...`
GloVe[6]	`| There | 's | ...`	`| That | 's | ...`
fastText[7]	`| There | 's | ...`	`| That's | ...`
Word2Vec[8]	`| There's | ...`	`| That's | ...`

Named Entity Recognition models. We select most hyperparameters based on previous work (Reimers and Gurevych, 2017), and adjust the optimizer hyperparameters based on its performance on the development set. Specifically, the outputs of language models are projected to 100 dimension, the decoding part is equipped with 256 hidden states in each direction, the optimization would be conducted with Nadam (Dozat, 2016), and the gradient is clipped at 1. During the training, dropouts with the probability 0.5 are applied to every layer, and the output of all representation modules would be further randomly dropped with the probability of 0.1. Since these two NER corpora are relative small, we would conduct the training of the final model on both the training set and the development set. The same strategy is also adopted for the training of baseline models.

4.4 Performance Comparison

We summarize the performance in Table 2. Specifically, we consider two variants of the proposed method: Neural-Char-CRF (NLTK) uses NLTK-based embedding alignment and Neural-Char-CRF (Match) uses string matching-based embedding alignment. We can observe a significant performance improvement of Neural-Char-CRF over baseline methods (by 3.70 and 6.65 absolute F_1 gain on TNT and BTC respectively), which verifies the effectiveness of our proposed framework. Further analyses are discussed as below.

Comparing String Matching with NLTK. We

[6]http://nlp.stanford.edu/data/glove.840B.300d.zip

[7]https://dl.fbaipublicfiles.com/fasttext/vectors-english/crawl-300d-2M.vec.zip

[8]https://github.com/loretoparisi/word2vec-twitter

Table 4: Model Performance with Perfect Tokenization ($\overline{\Delta}$ is the average performance difference to those with three system tokenizers in Table 2).

	TNT		BTC	
	F_1	$\overline{\Delta}$	F_1	$\overline{\Delta}$
TwitterNER	73.24	1.91	65.67	19.13
LSTM-CNN-CRF	81.74	9.70	66.48	25.02
LM-LSTM-CRF	81.91	9.83	69.29	26.67
Flair	84.34	10.23	71.41	26.77

noticed that models with string matching-based alignment achieves similar and even better performance than models with NLTK-based alignment. We conduct further analysis to get more insights about this phenomenon. As shown in Table 3, different pre-processing rules are applied in different embedding learning methods, and thus lead to different dictionaries. In previous NER methods or the toknenization induced word embedding approach, a simple Lookup Table query would be used to align the embedding dictionary with the pre-processed word sequence. This practice may lead to a non-optimal alignment, when the preprocessing fails to meet some properties of the pretrained embedding. For example, if the tokenizer used for NER input didn't separate "That" and "'s" when using the pre-trained GloVe embedding (Pennington et al., 2014), the input "That's" would be mapped to the "unknown token". At the same time, our string matching approach is self-adaptive and has the ability to align different embedding individually. Therefore, by leveraging our proposed string matching approach, Neural-Char-CRF has the potential to be tokenization-free and can be constructed with less human endeavor.

Importance of Pre-processing. One can easily observe that pre-processing plays a crucial role on the performance. For example, switching the tokenizer from NLTK to spaCy or Stanford CoreNLP leads to a drop of more than 20 absolute F_1 points. We further conduct experiments to verify our intuition to conduct NER in the raw-to-end manner. Specifically, we construct models with gold-tokenized training data and evaluate their performance with NLTK, which achieves the best performance among all three tokenizers. The results summarized in Table 4 show that, after replacing NLTK-tokenized training data with gold-tokenized training data, F_1 improves significantly. It verifies our intuition to build the raw-to-end framework.

Ablation Study. As shown in Table 5, we conduct experiments with our best performing model (*i.e.*, Neural-Char-CRF (Match)) to study the effect of the two representation modules. The first (referred as "– language model") excludes the contextualized character representation module from Neural-Char-CRF, the second (referred as "– string match") replaces the proposed two modules with static character embedding. The results demonstrate the effectiveness of both proposed representation modules – significant performance improvements can be observed by adapting either of these two approaches.

Table 5: Ablation Study on Representation Modules.

Method	TNT	BTC
Neural-Char-CRF (Match)	86.96	74.98
– language model	80.72	63.71
– string match	75.28	59.35

4.5 Case Studies

Table 6 shows the output of Neural-Char-CRF and a strong baseline Flair on the TNT dataset. As for the first sentence, since both spaCy and Stanford CoreNLP fail to detect the word boundary of "NBA", methods that rely on tokenization results cannot identify this entity successfully. At the same time, the second sentence exhibits that Neural-Char-CRF can effectively leverage the context information.

5 Related Work

5.1 Named Entity Recognition

Most NER systems, as mentioned before, are constructed as sequence labeling models. Typically, in traditional methods, handcrafted features are leveraged to capture textual signals, and conditional random fields (CRF) are employed to model label dependencies (Finkel et al., 2005; Settles, 2004; Leaman et al., 2008). Recent advances in neural models allow us to better represent natural language and freed domain experts from handcrafting features on many tasks. Bi-LSTM-CRF leverages both word embedding and handcrafted features, combines neural networks with CRF and shows improvements over previous methods (Huang et al., 2015); LSTM-CNN further incorporates CNN and illustrates the potential of capturing character-level signals (Chiu and Nichols, 2016); LSTM-CRF and LSTM-CNN-CRF are proposed to get rid of handcrafted features and demonstrate the feasibility to

Table 6: Case study on the TNT dataset. Incorrect outputs are marked as red and bold.

	LeBron named to 10th All-#NBA First Team	Moving on up: @GeeksOUT and @FLAMECON are now on display in Times Square.
Flair (NLTK)	<LeBron, PER> <NBA, ORG>	**<GeeksOUT, PER>** **<FLAMECON, PER>** <Times Squere, LOC>
Flair (spaCy)	<LeBron, PER> **<All-#NBA, O>**	**<@GeeksOUT, PER>** **<@FLAMECON, ORG>** <Times Squere, LOC>
Flair (Stanford CoreNLP)	<LeBron, PER> **<#NBA, ORG>**	**<@GeeksOUT, PER>** **<@FLAMECON, PER>** <Times Squere, LOC>
Neural-Char-CRF	<LeBron, PER> <NBA, ORG>	<GeeksOUT, ORG> <FLAMECON, ORG> <Times Squere, LOC>

fully rely on representation learning to capture the textual signals (Lample et al., 2016; Ma and Hovy, 2016). Meanwhile, CharNER was proposed to make initial predictions at the character-level, however, this method still relies on tokenization to further regularizes and refines model outputs to ensure that the detected word boundaries is maintained in its final predictions, and thus it can be viewed as a variant of the word-level model (Kuru et al., 2016). More recently, language modelings are noticed to be dramatically effective as the representation module for NER (Peters et al., 2017, 2018; Liu et al., 2018a,b; Akbik et al., 2018; Liu et al., 2019b). Comparing these methods, it is noticed that most improvements are brought by new representation techniques, which allows us to construct the model in a more data-driven manner. Despite the effectiveness of these methods, as discussed before, all of them rely on preprocessing components and would have deteriorated performance on noisy texts like social media. In this paper, we propose to conduct NER in a raw-to-end manner.

5.2 Word Embedding

Most word embedding methods are based on the distributional hypothesis, *i.e.*, "a word is characterized by the company it keeps" (Harris, 1954), and learn word representations by analyzing their contexts. Unlike the previous work (Bengio et al., 2003; Hinton, 1986), word2vec (Mikolov et al., 2013) leverages the hierarchical softmax and the negative sampling, thus can scale up to extensive corpora. Recent work shows that word embedding could cover textual information of various levels (Artetxe et al., 2018). It has been shown that, by properly handling such embedding, the model performance can be boosted significantly (Liu et al., 2019b; Lin et al., 2019). Instead of trusting the pre-processing results, we develop two strategies to build robust word embedding alignments that suffer less from pre-processing errors.

5.3 Contextualized Representations

By leveraging pre-trained machine translation models, CoVe constructs contextualized word representations (McCann et al., 2017). After that, ELMo replaces translation with language modeling, which does not require annotations and has nearly unlimited corpora (Peters et al., 2018). It demonstrates significant improvements on various NLP tasks. A more comprehensive comparison shows that, comparing to machine translation (Zhang and Bowman, 2018), language modeling is more effective as the pre-training task, even after limiting the size of its training data. At the same time, lots of attentions have been attracted to leveraging language modeling to build sentence representations (Howard and Ruder, 2018; Radford et al., 2018; Devlin et al., 2018). Besides word-level language models, character-level language models have also been leveraged to construct the contextualized representation (Liu et al., 2018b; Akbik et al., 2018). These models are designed to get the contextualized representation for a span of characters (i.e., words and sentences). Here, we present a contextualized character representation model, which provides context information at the character-level.

6 Conclusion and Future Work

In this paper, we study the Named Entity Recognition task in noisy, user-generated texts. Recognizing the importance of pre-processing, we propose a raw-to-end framework Neural-Char-CRF. It takes raw input as character sequences and makes end-to-end predictions, thus relying less on preprocessing and suffering less from error propagation. Two novel representation learning modules are tailored to better capture the textual signals. Empirical results on the two tweets datasets demonstrate the superior performance of the proposed Neural-Char-CRF method. Performance analysis, ablation study and case studies further verify our intuitions. In the future work, We plan to apply our method to other tasks, domains and languages.

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649.

Mikel Artetxe, Gorka Labaka, Inigo Lopez-Gazpio, and Eneko Agirre. 2018. Uncovering divergent linguistic information in word embeddings with lessons for intrinsic and extrinsic evaluation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 282–291.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Steven Bird, Ewan Klein, Edward Loper, and Jason Baldridge. 2008. Multidisciplinary instruction with the natural language toolkit. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*, pages 62–70. Association for Computational Linguistics.

Colin Cherry and Hongyu Guo. 2015. The unreasonable effectiveness of word representations for twitter named entity recognition. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 735–745.

Jason PC Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Leon Derczynski, Kalina Bontcheva, and Ian Roberts. 2016. Broad twitter corpus: A diverse named entity recognition resource. In *Proceedings of COLING 2016*, pages 1169–1179.

Leon Derczynski, Eric Nichols, Marieke van Erp, and Nut Limsopatham. 2017. Results of the wnut2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 140–147.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Timothy Dozat. 2016. Incorporating nesterov momentum into adam.

J. R. Finkel, T. Grenager, and C. Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Association for Computational Linguistics (ACL)*, pages 363–370.

Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

Geoffrey E Hinton. 1986. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. Version: 2.0.18.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 328–339.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Onur Kuru, Ozan Arkan Can, and Deniz Yuret. 2016. Charner: Character-level named entity recognition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 911–921.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of NAACL-HLT*, pages 260–270.

Robert Leaman, Graciela Gonzalez, et al. 2008. Banner: an executable survey of advances in biomedical named entity recognition. In *Pacific Symposium on Biocomputing*, volume 13, pages 652–663. Citeseer.

Ying Lin, Liyuan Liu, Heng Ji, Dong Yu, and Jiawei Han. 2019. Reliability-aware dynamic feature composition for name tagging. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 165–174.

Jialu Liu, Jingbo Shang, Chi Wang, Xiang Ren, and Jiawei Han. 2015. Mining quality phrases from massive text corpora. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1729–1744. ACM.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2019a. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.

Liyuan Liu, Xiang Ren, Jingbo Shang, Xiaotao Gu, Jian Peng, and Jiawei Han. 2018a. Efficient contextualized representation: Language model pruning for sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1215–1225.

Liyuan Liu, Jingbo Shang, and Jiawei Han. 2019b. Arabic named entity recognition: What works and whats next. In *Proceedings of the Fourth Arabic Natural Language Processing Workshop*, pages 60–67.

Liyuan Liu, Jingbo Shang, Xiang Ren, Frank Fangzheng Xu, Huan Gui, Jian Peng, and Jiawei Han. 2018b. Empower sequence labeling with task-aware neural language model. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Di Lu, Leonardo Neves, Vitor Carvalho, Ning Zhang, and Heng Ji. 2018. Visual attention model for name tagging in multimodal social media. In *Proceedings of the ACL*, volume 1, pages 1990–1999.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Shubhanshu Mishra and Jana Diesner. 2016. Semi-supervised named entity recognition in noisy-text. In *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, pages 203–212. The COLING 2016 Organizing Committee.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the ACL*, volume 1, pages 1756–1765.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, volume 1, pages 2227–2237.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the ACL*.

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

Burr Settles. 2004. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the international joint workshop on natural language processing in biomedicine and its applications*, pages 104–107. Association for Computational Linguistics.

Jingbo Shang, Jialu Liu, Meng Jiang, Xiang Ren, Clare R Voss, and Jiawei Han. 2018. Automated phrase mining from massive text corpora. *IEEE TKDE*.

Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21.

Benjamin Strauss, Bethany Toma, Alan Ritter, Marie-Catherine De Marneffe, and Wei Xu. 2016. Results of the wnut16 named entity recognition shared task. In *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, pages 138–144.

Kelly W Zhang and Samuel R Bowman. 2018. Language modeling teaches you more syntax than translation does: Lessons learned through auxiliary task analysis. *EMNLP 2018*, page 359.

A Benchmark Annotation Conversion

In our implementations, there are three stages for recovering the character-level annotations from word-level annotations. First, we applied several pre-processing pipelines to regularize the raw text (include html unescape and unicode normalize). Comparing their inputs and outputs, we can build a dictionary from the raw character sequences to the processed character sequences. Secondly, we revert this dictionary, remove all spaces in the original texts and directly concatenate all words in the tokenized sequence. Then we try to align these two strings through the dictionary built in the last step, and record this alignment as a index mapping. In the end, we would calculate entity positions based on the index mapping, and generate character-level annotations based on the BIOES schema.

B Pipeline Implementation Details

Stanford and spaCy tokenizer are used to detect the boundaries in the text by taking in raw text input and outputing a list of tokenized words. However, Stanford tokenizer applies transformation on certain characters, which makes the tokenized result different from the original text. For example, '(' is transformed into '**-LRB-**', and '**. .**' is transformed into '**...**' (quotes added for clarity). This causes problems in boundary detection as the tokenized words and original text do not match. We looked through all the data in **TNT** and **BTC** and collected a list of transformed words and their original text, see Table 6. The transferred words are mapped back to find the correct boundaries. Its worthwhile to mention that, we only map the tokenized words back to detect boundaries in original text and to compare predicted entities with gold-standard. When it comes to training the model and making predictions, we still use tokenized words without mapping.

Different text may be transformed to the same tokenized word, so we find the best mapping such that between all consecutive boundaries detected, only spaces or untokenizable characters remain.

Transformed	Original
``	" ' ' “ ” «
"	" » ”
-LRB-	(
-RRB-	)
-LSB-	[
-RSB-	]
-LCB-	{
-RCB-	}
...	… .. . .
`	' ‘ ’
--	– — ―
'	’

Figure 6: Table reflects the mapping from the original characters to their transformed characters by Stanford Tokenizer. In this process, different characters can be transformed to a same sequence of characters, and different sequences of characters may come from a same character.