Neural Named Entity Boundary Detection

Jing Li, Aixin Sun, and Yukun Ma

Abstract—In this paper, we focus on *named entity boundary detection*, which is to detect the start and end boundaries of an entity mention in text, without predicting its type. The detected entities are input to entity linking or fine-grained typing systems for semantic enrichment. We propose BDRYBOT, a recurrent neural network encoder-decoder framework with a pointer network to detect entity boundaries from a given sentence. The encoder considers both character-level representations and word-level embeddings to represent the input words. In this way, BDRYBOT does not require any hand-crafted features. Because of the pointer network, BDRYBOT overcomes the problem of variable size output vocabulary and the issue of sparse boundary tags. We conduct two sets of experiments, in-domain detection and cross-domain detection, on six datasets. Our results show that BDRYBOT achieves state-of-the-art performance against five baselines. In addition, our proposed approach can be further enhanced when incorporating contextualized language embeddings into token representations.

Index Terms—Named entity boundary detection, neural networks, pointer networks, feature engineering.

1 INTRODUCTION

Named-entity recognition (NER) is a fundamental task in natural language processing which aims at jointly resolving the boundaries and type of a named entity in text. In this paper, we ignore the entity typing and focus on the sub-task of *named entity boundary detection*, which is to detect the start and end boundaries of an entity mention in text. The sub-task is motivated by the following three key observations:

First, *fine-grained* entity typing systems, such as FIGER [1], FINET [2], AFET [3], and SANE [4], have recently gained significant research interests. However, many studies on fine-grained typing either manually label entity boundaries or assume that entity boundaries have already been pre-detected [2]. In addition, some studies utilize off-the-shelf NER systems to detect named entity boundaries. For example, AFET [3] uses DBpedia Spotlight3 to identify named entities. FINET [2] uses Stanford CoreNLP to identify entities and then makes fine-grained type predictions. However, off-the-shelf systems are not specifically designed for entity boundary detection. As a consequence, errors made in entity boundary detection inevitably mislead and adversely affect subsequent entity-typing systems. This has created an overwhelming demand for more accurate and robust boundary detection approaches.

Second, the availability of knowledge bases *e.g.*, Wikipedia, FreeBase, ProBase, enables the study of entity linking, which is to determine the identity of entities mentioned in text. Because of the overwhelming entity types defined in knowledge bases, coarse types defined in traditional NER systems become less necessary. Further, there could be conflicts between the types predicted by NER and the types of the entities disambiguated through the entity linking process.

- *J. Li is with the Inception Institute of Artificial Intelligence, United Arab Emirates. This work was done when the author was with Nanyang Technological University, Singapore. E-mail: jli030@e.ntu.edu.sg.*
- *A. Sun and Y. Ma are with Nanyang Technological University, Singapore 639798. E-mail: axsun@ntu.edu.sg; mayu0010@e.ntu.edu.sg.*

Third, boundary detection can be framed as a sequence labeling problem, where the task is to predict a sequence of 'yes/no' boundary tags at word level in a sentence. Recent neural sequence labeling approaches widely fall into two categories: recurrent neural networks with conditional random fields output layer (RNN-CRF) and recurrent neural networks with recurrent neural networks (RNN-RNN)[1]. However, RNN-CRF suffers from the issue of sparse boundary tags, because entity tokens are rare and non-entity tokens are common in a typical sentence [5]. On the other hand, although RNN-RNN [6] can handle input/output sequences of varying lengths, the output vocabulary (from which the output sequence is drawn) on the top of decoder RNN is fixed. Consequently, different models need to be trained with respect to different output vocabularies.

To alleviate the above issues, we propose BDRYBOT, a neural model for entity boundary detection. BDRYBOT adopts a sequence to sequence model with RNN networks and the *pointer* mechanism [7]. Specifically, to encode a sentence, the model employs a bidirectional RNN to model sequential dependencies. Each word in the given sentence is represented by both its word-level embeddings and character-level representations learned by a sliding convolutional neural network. The decoder is an unidirectional RNN, and the entity boundaries are inferred by using the *pointer* mechanism. In this way, BDRYBOT effectively handles variable sized vocabulary in the output to produce entity boundaries based on input sequence. In summary, we make three contributions:

- We propose BDRYBOT, an end-to-end model for entity boundary detection. BDRYBOT learns informative features automatically while alleviating the problem of tag sparsity in output sequence and the problem of variable size output vocabulary.
- We conduct both in-domain and cross-domain evaluations on six datasets. Our results show that BDRYBOT achieves state-of-the-art results against five

1. One RNN to encode and one RNN as a language model to generate the output sequence.

baselines and can be further augmented by recent pre-trained language models.
- We make BDRYBOT available online[2] (trained on CoNLL2003) and provide users with Application Programming Interface (API).

2 Related Work

Named Entity Recognition. There are three common paradigms for NER [8]: *knowledge-based unsupervised*, *feature-based supervised* and *neural-based* systems. *Knowledge-based unsupervised* systems rely on lexical knowledge, such as domain-specific gazetteers and shallow syntactic knowledge. These systems work very well only when there is exhaustive lexicon. *Feature-based supervised* systems cast NER as a multi-class classification or sequence labeling task. Feature engineering is a critical step in these systems.

Because neural-based systems have the advantage of inferring latent features and learning sequence labels in an end-to-end fashion, many neural architectures have been widely applied in NER. Collobert *et al.* [9] first adopted neural models in NER, where an architecture based on temporal convolutional neural networks (CNNs) over a word sequence was proposed. Since then, there has been a growing body of work on neural-based NER. Existing neural-based systems can be unified into a framework with three components: an input representation, context encoder and tag decoder. Commonly used input representations include word-level and character-level representations [10]. Widely used context encoder architectures include CNNs [9], recurrent neural networks (RNNs) [11], recursive neural networks [12] and deep transformers [13]. At the top of the context encoder, a CRF layer [11], a pointer network [14], or an RNN layer [6] is employed to make sequence label predictions.

Named Entity Typing. Named entity typing is the task for assigning types or labels such as organization, location to the detected entity mentions in a document. Existing studies can be categorized into *coarse-grained* typing and *fine-grained* typing. Coarse-grained typing focuses on a small set of coarse types, such as *person*, *location*, *organization* and *misc*. Fine-grained typing focuses on a much larger set of fine-grained types organized in a tree-structured hierarchy. Lin *et al.* [15] proposed a fine-grained system, which propagates over 1,000 types from linked Wikipedia entities to unlinkable entities. Context-aware systems (*e.g.,* FINET [2] and SANE [4]), embedding-based systems (*e.g.,* FIGER [1], AFET [3]), and partial-label system (*e.g.,* PLE [16]), are recently been developed to address fine-grained entity typing.

Named entity typing systems assume that named entities have already been detected in documents. In our task, the output of entity boundary detection therefore provides input to named entity typing systems. The detected mentions can also be input to entity linking systems.

3 Named Entity Boundary Detection

Figure 1(a) shows the model architecture of BDRYBOT. Given a sentence, we represent each word with a distributed representation based on its character-level and word-level embeddings. Then we use a bidirectional recurrent neural network to capture syntactic and lexical information in the sentence. Finally, we use a pointer mechanism to infer entity boundaries based on the hidden states of decoder.

3.1 Input Representation Phase

The input representation in our model consists of character-level representation and word-level representation. Previous studies [11], [17] have shown that character-level information (*e.g.,* prefix and suffix of a word) is an effective resource for NER task. Two kinds of network structures have been used to extract character-level representation, *i.e.,* CNN and BiLSTM. In our model, we use CNN because of its lower computational cost. Our design is similar to [17], and the main difference is that we use a sliding convolution layer (*i.e.,* variable window size of convolution filters) to capture character n-grams in a word, shown in Figure 1(b).

Given an input sentence $\boldsymbol{W} = (w_1, w_2, \ldots, w_N)$ of length N. Let w_n denote its n-th word. The character-level representation and word-level embedding (*e.g.,* pretrained embedding) for w_n are concatenated as its final representation, $\boldsymbol{x}_n \in \mathbb{R}^K$, where K represents dimension of $\boldsymbol{x}_n$.

3.2 Encoding Phase

We encode the input sequence $\mathbf{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N)$ using a RNN. With hidden cells like long short-term memory (LSTM) and gated recurrent unit (GRU), RNN captures long distance dependencies without running into the problems of gradient vanishing or explosion. We use GRU which is computationally cheaper than LSTM. Specifically, GRU activations at time step n are computed as follows:

$$\boldsymbol{z}_n = \sigma(\boldsymbol{U}_z \boldsymbol{x}_n + \boldsymbol{R}_z \boldsymbol{h}_{n-1} + \boldsymbol{b}_z) \quad (1)$$

$$\boldsymbol{r}_n = \sigma(\boldsymbol{U}_r \boldsymbol{x}_n + \boldsymbol{R}_r \boldsymbol{h}_{n-1} + \boldsymbol{b}_r) \quad (2)$$

$$\boldsymbol{n}_n = \tanh(\boldsymbol{U}_h \boldsymbol{x}_n + \boldsymbol{R}_h(\boldsymbol{r}_n \odot \boldsymbol{h}_{n-1}) + \boldsymbol{b}_h) \quad (3)$$

$$\boldsymbol{h}_n = \boldsymbol{z}_n \odot \boldsymbol{h}_{n-1} + (1 - \boldsymbol{z}_n) \odot \boldsymbol{y}_n \quad (4)$$

where $\sigma()$ is sigmoid function, $\tanh()$ is the hyperbolic tangent function, $\odot$ is element-wise multiplication, $\boldsymbol{z}_n$ is the update gate vector, $\boldsymbol{r}_n$ is the reset gate vector, $\boldsymbol{n}_n$ is the new gate vector, and $\boldsymbol{h}_n$ is the hidden state at time step n. $\boldsymbol{U}$, $\boldsymbol{R}$, $\boldsymbol{b}$ are the parameters of the encoder that need to learn.

We use a bi-directional GRU (BiGRU) network to memorize past and future information in the input sequence. Each hidden state of BiGRU is formalized as:

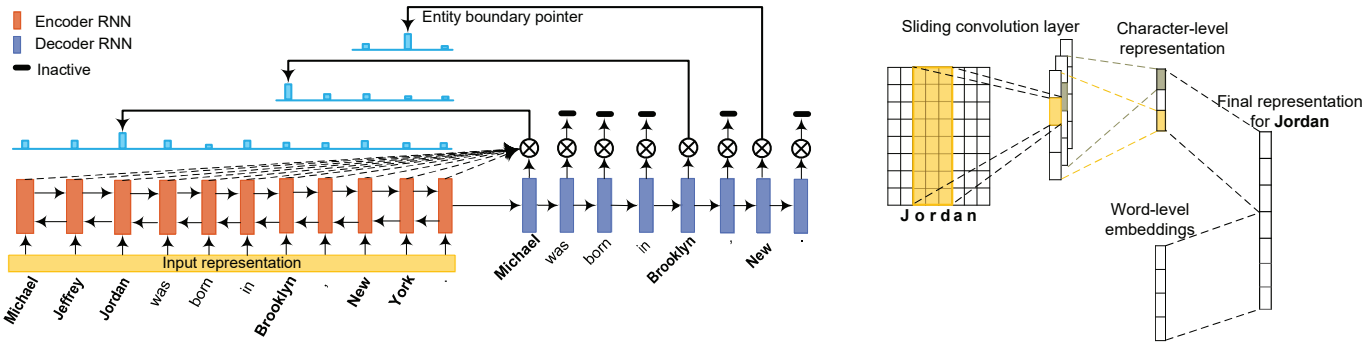
$$\boldsymbol{h}_n = \overrightarrow{\boldsymbol{h}}_n \oplus \overleftarrow{\boldsymbol{h}}_n \quad (5)$$

where $\oplus$ indicates concatenation operation, $\overrightarrow{\boldsymbol{h}}_n$ and $\overleftarrow{\boldsymbol{h}}_n$ are hidden states of the forward (left-to-right) and backward (right-to-left) GRUs, respectively. Assuming the size of the GRU is H, the encoder yields hidden states in $\boldsymbol{h} \in \mathbb{R}^{N \times 2H}$.

3.3 Decoding Phase

Because the number of boundaries in output vary with the given input, it is natural to use RNN-based models to decode the output. At each step, the decoder takes a word w_n from the input sequence as input, and transforms it to its

2. http://138.197.118.157:8000/bdrybot/



(a) The architecture of BDRYBOT. (b) Representation of an input word "*Jordan*".

Fig. 1. Model architecture with input sentence: "*Michael Jeffrey Jordan was born in Brooklyn, New York.*"

distributed representation $\boldsymbol{x}_n$ by looking up the corresponding embedding matrix. It then passes $\boldsymbol{x}_n$ through a GRU-based unidirectional hidden layer. The decoder hidden state at time step m is computed by:

$$\boldsymbol{d}_m = GRU(\boldsymbol{x}_m, \boldsymbol{\theta}) \tag{6}$$

where $\boldsymbol{\theta}$ are the parameters in the hidden layer of the decoder, which have the same form as defined in Equations (1) – (4). Note that, not every word from the input sentence needs pass to decoder. Shown in Figure 1(a), "*Jordan*" is the end boundary of mention "*Michael Jeffrey Jordan*", so the two words "*Jeffrey*" and "*Jordan*" will not be passed to decoder. Suppose there are M time steps in decoder, the decoder produces hidden states in $\boldsymbol{d} \in \mathbb{R}^{M\times 2H}$ with $2H$ being the dimensions of the hidden layer of decoder. Again, the encoder is bidirectional (hidden size H), and the decoder is unidirectional (hidden size $2H$).

3.4 Pointing Phase

In the pointing phase, BDRYBOT detects entity boundaries only if the current input is a start boundary. Otherwise, it will switch decoder status to *inactive* and no boundary will be detected. Consider the input shown in Figure 1(a). Decoder starts with input w_1, *i.e., Michael*, which is a start of an entity mention in this example. Thus, BDRYBOT computes an output distribution over all positions (w_1 to w_{11}) in the input sentence. w_3 (*Jordan*) is identified as the end boundary of w_1 (*Michael*). Then, w_4 is taken as the input, the status of decoder switches to *inactive* because "*was*" is not a start boundary of any entity mention. Observe that, unlike traditional seq2seq models (*e.g.,* the ones used in neural machine translation), where the output vocabulary is fixed, the number of possible positions in the input sequence changes at each decoding step in BDRYBOT.

In order to achieve this mechanism as described above, we pad the hidden states of encoder with a sentinel word representing *inactive*. That is, the decoder should point this sentinel word once the current input is not a start boundary of an entity. We also extend pointer network [7] with a direction-aware mechanism. Recall that $\boldsymbol{h} \in \mathbb{R}^{N\times 2H}$ and $\boldsymbol{d} \in \mathbb{R}^{M\times H}$ are the hidden states in encoder and decoder, respectively. We first pad $\boldsymbol{h}$ with a sentinel vector as follows:

$$\boldsymbol{h} = [\boldsymbol{h}; 0] \tag{7}$$

where $\boldsymbol{h} \in \mathbb{R}^{(N+1)\times 2H}$. Then, we use an attention mechanism to compute the distribution of end boundary over all possible positions in the input sequence at decoding step m:

$$u_j^m = \boldsymbol{v}^T \tanh(\boldsymbol{G}_1\boldsymbol{h}_j + \boldsymbol{G}_2\boldsymbol{d}_m), \quad \text{for } j \in (m, \ldots, M) \tag{8}$$

$$p(y_m|\boldsymbol{x}_m) = \text{softmax}(\boldsymbol{u}^m) \tag{9}$$

Here, $\boldsymbol{v}$, $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ are learnable parameters, $j \in [m, M]$ indicates a possible position in the input sequence, and *softmax* normalized u_j^m indicating the probability that word w_j is an end boundary, given the start boundary w_m. When w_m is not a start boundary of any entity, the pointer is trained to point to the padded word w_{N+1}, *i.e., inactive*. For example, BDRYBOT points to "*inactive*" when given "*was*" as the decoder input in Figure 1(a).

3.5 Model Training

We use "*teacher forcing*" to train our model by supplying the ground-truth start units to the decoder RNN [18]. This mechanism forces the RNNs to stay close to the ground-truth start units and entity boundaries. The loss function $\mathcal{L}$ is the negative log likelihood of boundary distribution over the whole training set $\mathcal{D}$:

$$\mathcal{L}(\boldsymbol{\omega}) = \sum_{\mathcal{D}} \sum_{m=1}^{M} -\log p(y_m|\boldsymbol{x}_m; \boldsymbol{\omega}) + \frac{\lambda}{2}||\boldsymbol{\omega}||_2^2 \tag{10}$$

where $\boldsymbol{\omega}$ are the trainable parameters of the model, and λ is the strength of L_2 regularization.

When using the RNN decoder for prediction on test instance, the ground-truth boundaries are not available. Similar to traditional seq2seq decoders, we feed the input symbols based on the decoded symbol from the previous step, *e.g.,* we feed "*was*" after predicting a boundary at "*Jordan*" in Figure 1(a).

4 EXPERIMENTS

4.1 Experimental Setup

Datasets. We use five popular benchmark datasets and one dataset annotated by ourselves (*i.e.,* Yahoo!), to ascertain the effectiveness of our proposed approach. Because our task is boundary detection, we ignore entity types in all datasets.

- **CoNLL2003** - The CoNLL2003 NER Shared Task dataset is a collection of Reuters newswire articles

TABLE 1
Statistics of datasets.

Dataset	# Sentences			#Mentions
	Train	Dev	Test	
CoNLL2003	14,987	3,466	3,684	34,841
OntoNotes5.0	59,917	8,528	8,262	71,031
WikiGold	144,342	-	1,696	298,961
WNUT2017	3,394	1,009	1,287	3,890
BBN	32,739	-	6,338	79,730
Yahoo!	-	-	500	981

that contains a large portion of sports news [19]. It is annotated with four entity types.

- **OntoNotes5.0** - This dataset includes text from five different text genres: newswire, magazine, broadcast news, broadcast conversation, web data [20]. It is annotated with 18 entity types.
- **WikiGold** - It is a set of Wikipedia articles manually annotated with CoNLL2003 entity types [21]. The articles were randomly selected from a 2008 English dump and cover a number of topics.
- **WNUT2017** - It is a set of noisy user-generated text including Twitter, Reddit, YouTube comments, and StackExchange posts [22]. The aim of this dataset is to identify unusual, previously-unseen entities in the context of emerging discussions.
- **BBN** - This dataset consists of articles from Wall Street Journal [3]. The test dataset is manually annotated using 93 types. However, the training set is automatically annotated by DBpedia Spotlight3.
- **Yahoo!** - This dataset consists of 500 sentences collected from Yahoo! News comments. It is manually annotated with CoNLL2003 entity types.

For CoNLL2003, OntoNotes5.0, and WNUT2017, we follow the standard data splits. For WikiGold, we randomly sampled 10% of the training set to form a development set, on which we tune model parameters. Both BBN and Yahoo! datasets are used only for cross-domain evaluation. That is, we do not train our models on these two datasets, because BNN training set is noisy and Yahoo! dataset is very small. The statistics of datasets are reported in Table 1.

Metrics. We measure Precision (P), Recall (R), and F-score ($F1$) to evaluate entity boundary detection accuracy. Precision is the percentage of entities detected by the model that are correct. Recall is the percentage of entities in the dataset that should be detected by the model. A named entity is correctly detected only if it is *an exact match* of the corresponding entity in the ground-truth (ignoring entity type). Let g be the total number of annotated entities in ground-truth, h be the total number of entities detected by a model, and c be the total number of correctly detected entities by the model. Then $P = \frac{c}{h}$, $R = \frac{c}{g}$, and $F1 = \frac{2c}{g+h}$.

Baselines. We evaluate the proposed BDRYBOT against following competitors[3].

- **RegEx** - This rule-based model is created with regular expressions, based on word surface patterns, *e.g.*, letter case.
- **Shallow-CRF** - This model trains a CRF using the commonly used token-level features [23]. All features can be easily extracted without complicated computation.
- **StanfordNER** - This CRF model is the Stanford Named Entity Recognizer version 3.9.1[4].
- **BiLSTM-CRF** - This model [17] utilizes BiLSTM to encode word sequence, and CRF to infer decoder tags. Both BiLSTM-CRF and BDRYBOT use the input representation illustrated in Figure 1(b).
- **BiLSTM-Softmax** - This model uses the Multi-Layered Perceptron and Softmax layers to infer decoder tags instead of CRF as in BiLSTM-CRF.
- **Pre-trained Language Models** - These models pre-train language models on large corpora. We build up a softmax layer on the top of ELMo [10] or BERT [13]. ELMo-FineTune and BERT-FineTune are both fine-tuned on training datasets and finally evaluated on test datasets. Meanwhile, we also consider the BERT embedding as a part of token representation in our approach and BERT is fine-tuned during training.

Implementation Details. For all neural models, we use GloVe 300-dimensional embeddings released by Stanford[5], which are fine-tuned during training. We first use a grid search strategy to tune hyper-parameters on CoNLL2003 and then fine-tune the parameters on other datasets.

The dimension of character-level representation is 100 and the CNN sliding windows of filters are $[2, 3, 4, 5]$. The total number of CNN filters is 100. The bidirectional encoder GRU each has depth of 3 and hidden size 200. Each decode GRU has depth of 3 and hidden size 400. Noted that the encoder GRU is bidirectional and decoder GRU is unidirectional in our model. Thus, the decoder is twice the hidden size of encoder. The Adam optimizer was adopted with a learning rate 0.001 selected from {0.01, 0.001, 0.0001}. We use a dropout of 0.5 after the convolution or recurrent layers. The decay rate is 0.09 and the gradient clip is 5.0. For neural baselines, we use exactly the same hyper-parameters grid and training procedure as our proposed model above. We report the results based on the best performance on the development set. All neural network models are implemented with PyTorch framework and evaluated on NVIDIA Tesla P100 GPUs.

4.2 In-domain Evaluation

Table 2 reports the results of all models on CoNLL2003, OntoNotes5.0, WikiGold, and WNUT2017. From the results, we make the following observations:

First, our BDRYBOT model outperforms all baselines in terms of R and $F1$ on all datasets. In terms of $F1$ score, BDRYBOT is better than the strong baseline BiLSTM-CRF on CoNLL2003 (improv. 0.85%), OntoNotes5.0 (improv. 1.74%), WikiGold (improv. 1.50%) and WNUT2017 (improv. 1.88%). We attribute this to the fact that BDRYBOT can effectively capture the dependencies of input sentence when the entity boundaries are sparse. Our solution also provides a new

3. Both our approach and all competitors utilize the BIOES tag schema. BIOES stands for Begin, Inside, Outside, End, Single.

4. https://nlp.stanford.edu/software/CRF-NER.html

5. http://nlp.stanford.edu/projects/glove/

TABLE 2
Boundary detection accuracy on four benchmark datasets. Best results are in bold and significant improvements of F1 over the underlined neural methods (BiLSTM-Softmax and BiLSTM-CRF) are marked with * (p-value < 0.05).

Method	CoNLL2003			OntoNotes5.0			WikiGold			WNUT2017		
	P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$
RegEx	0.662	0.880	0.756	0.502	0.675	0.576	0.576	0.719	0.640	0.273	0.557	0.366
Shallow-CRF	0.934	0.904	0.919	0.907	0.858	0.882	0.837	0.828	0.833	**0.722**	0.120	0.206
StanfordNER	0.939	0.928	0.933	0.780	0.776	0.778	0.831	0.806	0.818	0.523	0.455	0.486
BiLSTM-Softmax	0.943	0.955	0.949	0.911	0.927	0.919	0.849	0.875	0.862	0.563	0.583	0.573
BiLSTM-CRF	0.946	0.956	0.951	0.916	0.928	0.922	0.848	0.891	0.869	0.541	0.637	0.585
BDRYBOT (ours)	**0.956**	**0.962**	**0.959***	**0.933**	**0.943**	**0.938***	**0.853**	**0.913**	**0.882***	0.553	**0.647**	**0.596***
ELMo-FineTune	0.958	**0.972**	0.965	0.938	0.954	0.946	0.903	**0.921**	0.912	0.597	0.653	0.624
BERT-FineTune	0.973	0.965	0.969	**0.963**	0.941	0.952	0.922	0.914	0.918	0.620	0.642	0.631
BDRYBOT +BERT	**0.979**	0.969	**0.974**	0.960	**0.956**	**0.958**	**0.931**	0.915	**0.923**	**0.624**	**0.692**	**0.656**

TABLE 3
Cross-domain evaluation results. $\star$ denotes that the model does not require training, is directly evaluated on test sets. Best results are in bold and significant improvements of F1 over the underlined neural methods (BiLSTM-Softmax and BiLSTM-CRF) are marked with * (p-value < 0.05).

Method	BBN (CoNLL2003)			Yahoo! (CoNLL2003)			BBN (OntoNotes5.0)			Yahoo! (OntoNotes5.0)		
	P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$
RegEx$\star$	0.475	0.658	0.552	0.432	0.641	0.516	0.475	0.658	0.552	0.432	0.641	0.516
Shallow-CRF	0.622	0.696	0.657	0.668	0.701	0.684	0.678	0.742	0.708	**0.748**	0.579	0.653
StanfordNER$\star$	0.706	0.819	0.758	0.737	0.717	0.727	0.706	0.819	0.758	0.737	0.717	0.727
BiLSTM-Softmax	0.717	0.857	0.781	0.667	0.804	0.729	0.722	0.839	0.776	0.701	0.775	0.736
BiLSTM-CRF	0.722	0.865	0.787	0.666	0.817	0.734	0.726	0.845	0.781	0.705	0.783	0.742
BDRYBOT (ours)	**0.727**	**0.874**	**0.794***	0.684	**0.821**	**0.746***	**0.731**	**0.864**	**0.792***	0.720	**0.789**	**0.753***
ELMo-FineTune	0.746	0.891	0.812	0.698	0.838	0.762	0.767	0.883	0.821	0.734	0.814	0.772
BERT-FineTune	0.769	0.904	0.831	**0.719**	0.827	0.769	0.788	0.897	0.839	**0.767**	0.831	0.781
BDRYBOT +BERT	**0.785**	**0.912**	**0.844**	0.714	**0.845**	**0.774**	**0.796**	**0.912**	**0.850**	0.752	**0.839**	**0.793**

perspective to model sequence labeling task using pointer network, instead of the classic CRF-based approach.

Additionally, BDRYBOT has balanced precision and recall on CoNLL2003 and OntoNotes5.0. The recall is much higher than precision on WikiGold and WNUT2017. For Shallow-CRF and StanfordNER, precision is higher than recall on all datasets. Although Shallow-CRF achieves the best precision on WNUT2017, its recall is very low. We observe all models have low $F1$ scores on WNUT2017, because the dataset is collected for detecting emerging and rare entities from user-generated text. This is a challenging task and the best $F1$ of NER task on this dataset is 0.418[6]. Without considering entity typing, our model achieves $F1$ of 0.596.

Finally, compared to static word embeddings (*i.e.*, GloVe), fine-tuning pre-trained contextualized language models can significantly improve the performance. This is because these pre-trained models are commonly trained on large-scale corpora and the model parameters have stored vast amount of linguistic knowledge. The experimental results show that our approach can be further augmented when incorporating contextualize language models.

4.3 Cross-Domain Evaluation

We conduct a cross-domain evaluation of all models on two test sets (BBN and Yahoo!). Table 3 reports the results obtained by training the models on CoNLL2003 and OntoNotes5.0, and testing on BBN and Yahoo!. The models differ in their ability to adapt to new datasets. BDRYBOT outperforms all baselines on BBN and Yahoo in terms of R and $F1$. When trained on either CoNLL2003 or OntoNotes5.0, BDRYBOT achieves comparable performance on BBN test set. On Yahoo! test set, BDRYBOT trained on OntoNotes5.0 is slightly better than the version trained on CoNLL2003.

6. https://noisy-text.github.io/2017/emerging-rare-entities.html

As expected, the performance on cross-domain test sets is much worse than that on in-domain test sets. For example, when trained on CoNLL2003, BDRYBOT delivers $F1$ score of 0.959 on CoNLL2003 test set, and only 0.794 on BBN test set. The drop may be attributed to two reasons. One is that cross-domain test sets have different name entity distributions from training set. The other is that neural-inferring features from training set cannot be effectively transferred to cross-domain test set without any domain-adaptation. There is a possibility to improve the cross-domain performance with adversarial domain-adaptation techniques.

4.4 Qualitative Analysis

We show some example sentences from CoNLL2003 test set in Table 4. From positive examples, we observe that BDRYBOT successfully detects multiple-word entities like "*Bill Jordan*", "*International Confederation of Free Trade Unions*", "*Svetlana Gladishiva*" and "*World Cup Super G*". The negative examples suggest that our model could be misled by capitalization, like the wrongly detection of "*Group A*" in sentence N1. In sentence N2, the annotated ground-truth is "SKIING-WORLD CUP", while BDRYBOT considers "*FREESTYLE*" as a part of the entity. In the third negative example, our approach detects "*Sheffield*" instead of "*Sheffield Wednesday*". Note that, our model is not trained to detect time expressions, therefore the time entities "*Friday*" and "*Saturday*" are not detected in the sentence.

For better understanding, we visualize pointer attention weights of the first positive example in Figure 2. The words in y-axis are the input of GRU decoder. The words in x-axis are the input of GRU encoder. For example, for a given input "*International*", BDRYBOT detects the end boundary of this entity at the position "*Unions*". For a given input "*is*", BDRYBOT determines that it is not a start of an entity mention

TABLE 4
Positive and negative examples for named entity boundary detection by BDRYBOT.

Positive Examples:		
P1.	Bill Jordan is the general secretary of the International Confederation of Free Trade Unions.	
P2.	Relations between Clarke , Major good - spokesman.	
P3.	Svetlana Gladishiva of Russia won the women's World Cup Super G race on Saturday.	
Negative Examples:		**Ground-truth**
N1.	Hosts UAE play Kuwait and South Korea take on Indonesia on Saturday in **Group A** matches.	Group A is not an entity
N2.	**FREESTYLE SKIING-WORLD CUP** MOGUL RESULTS.	SKIING-WORLD CUP
N3.	Dutch forward Reggie Blinker had his indefinite suspension lifted by FIFA on Friday and was set to make his **Sheffield** Wednesday comeback against Liverpool on Saturday.	Sheffield Wednesday

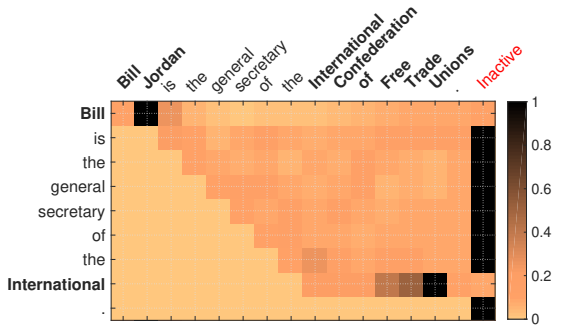


Fig. 2. Visualization of pointer attention weights for sentence "***Bill Jordan** is the general secretary of the **International Confederation of Free Trade Unions***.". Our model successfully detects the end boundaries at the positions: *Jordan* and *Unions*. The interactive visualization is available online http://138.197.118.157:8000/bdrybotapi/

by the sentinel word "*Inactive*". Observe that the identified boundaries have dominant attention weights, which implies that BDRYBOT can successfully learn sentence features for entity boundary detection.

5 CONCLUSION

In this paper, we propose BDRYBOT, an end-to-end neural model to detect entity boundaries from text. Because our model utilize a pointer network, it effectively addresses the issue of boundary tag sparsity. More importantly, compared with existing encoder-decoder models, BDRYBOT has the key advantage of inherently handling variable size output vocabulary. Through two sets of experiments, on in-domain detection and cross-domain detection, we demonstrate the effectiveness of BDRYBOT against state-of-the-art solutions on different datasets.

We consider two potential directions for future works. First, cross-domain adaptation remains challenging based on our experimental findings. Leveraging adversarial transfer learning and meta-learning to improve cross-domain adaptation is a very interesting direction. Second, improving text understanding and enhancing language representations using entity-level information is also a potential direction.

REFERENCES

[1] A. Abhishek, A. Anand, and A. Awekar, "Fine-grained entity type classification by jointly learning representations and label embeddings," in *EACL*, 2017, pp. 797–807.
[2] L. d. Corro, A. Abujabal, R. Gemulla, and G. Weikum, "Finet: Context-aware fine-grained named entity typing," in *EMNLP*, 2015, pp. 868–878.
[3] X. Ren, W. He, M. Qu, L. Huang, H. Ji, and J. Han, "Afet: Automatic fine-grained entity typing by hierarchical partial-label embedding," in *EMNLP*, 2016, pp. 1369–1378.
[4] A. Lal, A. Tomer, and C. R. Chowdary, "Sane: System for fine grained named entity typing on textual data," in *WWW*, 2017, pp. 227–230.
[5] S. Joty, G. Carenini, and R. T. Ng, "Codra: A novel discriminative framework for rhetorical analysis," *Computational Linguistics*, vol. 41, no. 3, pp. 385–435, 2015.
[6] Y. Shen, H. Yun, Z. C. Lipton, Y. Kronrod, and A. Anandkumar, "Deep active learning for named entity recognition," in *ICLR*, 2018.
[7] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NIPS*, 2015, pp. 2692–2700.
[8] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *CoRR*, vol. abs/1812.09449, 2018.
[9] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, 2011.
[10] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *NAACL-HLT*, 2018, pp. 2227–2237.
[11] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.
[12] P. Li, R. Dong, Y. Wang, J. Chou, and W. Ma, "Leveraging linguistic structures for named entity recognition with bidirectional recursive neural networks," in *EMNLP*, 2017, pp. 2664–2669.
[13] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019, pp. 4171–4186.
[14] F. Zhai, S. Potdar, B. Xiang, and B. Zhou, "Neural models for sequence chunking." in *AAAI*, 2017, pp. 3365–3371.
[15] T. Lin, O. Etzioni *et al.*, "No noun phrase left behind: detecting and typing unlinkable entities," in *EMNLP*, 2012, pp. 893–903.
[16] X. Ren, W. He, M. Qu, C. R. Voss, H. Ji, and J. Han, "Label noise reduction in entity typing by heterogeneous partial-label embedding," in *SIGKDD*, 2016, pp. 1825–1834.
[17] J. P. Chiu and E. Nichols, "Named entity recognition with bidirectional lstm-cnns," in *TACL*, vol. 4, 2016, pp. 357–370.
[18] A. M. Lamb, A. G. A. P. GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," in *NIPS*, 2016, pp. 4601–4609.
[19] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," in *NAACL-HLT*, 2003, pp. 142–147.
[20] S. Pradhan, A. Moschitti, N. Xue, O. Uryupina, and Y. Zhang, "Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes," in *EMNLP*, 2012, pp. 1–40.
[21] D. Balasuriya, N. Ringland, J. Nothman, T. Murphy, and J. R. Curran, "Named entity recognition in wikipedia," in *ACL-IJCNLP*, 2009, pp. 10–18.
[22] L. Derczynski, E. Nichols, M. van Erp, and N. Limsopatham, "Results of the wnut2017 shared task on novel and emerging entity recognition," in *W-NUT*, 2017, pp. 140–147.
[23] W. Liao and S. Veeramachaneni, "A simple semi-supervised algorithm for named entity recognition," in *NAACL-HLT*, 2009, pp. 58–65.

```
@article{li21bdrybot,
author    = {Jing Li and Aixin Sun and Yukun Ma},
title     = {Neural Named Entity Boundary Detection},
journal   = {IEEE Transactions on Knowledge and Data Engineering (TKDE)},
volume    = {33},
number    = {4},
pages     = {1790--1795},
year      = {2021},
url       = {https://doi.org/10.1109/TKDE.2020.2981329},
doi       = {10.1109/TKDE.2020.2981329},
}
```