Code-Switched Named Entity Recognition with Embedding Attention

Changhan Wang[†], **Kyunghyun Cho**[†‡] **and Douwe Kiela**[†]
[†] Facebook AI Research; [‡] New York University
{changhan,kyunghyuncho,dkiela}@fb.com

Abstract

We describe our work for the CALCS 2018 shared task on named entity recognition on code-switched data. Our system ranked first place for MS Arabic-Egyptian named entity recognition and third place for English-Spanish.

1 Introduction

The tendency for multilingual speakers to engage in code-switching—i.e, alternating between multiple languages or language varieties—poses important problems for NLP systems: traditional monolingual techniques quickly break down with input from mixed languages. Even for problems such as POS-tagging and language identification, which the community often considers "solved", performance deteriorates proportional to the degree of code-switching in the data. The shared task for the third workshop on Computational Approaches on Linguistic Code-Switching concerned named entity recognition (NER) for two code-switched language pairs (Aguilar et al., 2018): Modern Standard Arabic and Egyptian (MSA-EGY); and English-Spanish (ENG-SPA). Here, we describe our work on the shared task.

Traditional NER systems used to rely heavily on hand-crafted features and gazetteers, but have since been replaced by neural architectures that combine bidirectional LSTMs and CRFs (Lample et al., 2016). Equipped with supervised character-level representations and pre-trained unsupervised word embeddings, such neural architectures have not only come to dominate named entity recognition, but have also successfully been applied to code-switched language identification (Samih et al., 2016), which makes them highly suitable for the current task as well.

In this paper, we exploit recent advances in neural NLP systems, tailored to code-switching. We use high-quality FastText embeddings trained on Common Crawl (Grave et al., 2018; Mikolov et al., 2018) and employ shortcut-stacked sentence encoders (Nie and Bansal, 2017) to obtain deep token-level representations to feed into the CRF. In addition, we make use of an embedding-level attention mechanism that learns task-specific attention weights for multilingual and character-level representations, inspired by context-attentive embeddings (Kiela et al., 2018). In what follows, we describe our system in detail.

2 Approach

The input data consists of noisy user-generated social media text collected from Twitter. Code-switching can occur between different tweets in the training data, with many tweets being monolingual, but can also occur within tweets (e.g. "*[USER]: en los finales be like [URL]*") or even morphologically within words (e.g. "*pero esta twitteando y pitchandome los textos*"). The goal is to predict the correct IOB entity type for the following categories:

- [BI]-PER: Person
- [BI]-LOC: Location
- [BI]-ORG: Organization
- [BI]-GROUP: Group
- [BI]-TITLE: Title
- [BI]-PROD: Product
- [BI]-EVENT: Event
- [BI]-TIME: Time
- [BI]-OTHER: Other
- O: Any other token that is not an NE

The train/valid/test split for MSA-EGY was 10102/1122/1110. The train/valid/test split for ENG-SPA was 50757/832/15634.

The first work to combine CRFs with modern neural representation learning for NER is, to our knowledge, by Collobert et al. (2011). Our architecture is similar to more recent neural architectures for NER, e.g. Huang et al. (2015); Lample et al. (2016); Ma and Hovy (2016). Instead of using a straightforward bidirectional LSTM (BiLSTM), we use several layers and add shortcut connections. Instead of simply feeding in word (and/or character) embeddings, we add a self-attention mechanism.

2.1 Embedding Attention

We represent the input tweets on the word level and character level. For all available words in the data, we obtained FastText embeddings trained on Common Crawl and Wikipedia[1] for each language. For every word, we try to find an exact match in the FastText embeddings, or if that is not available we check if it is present in lower case. When a word embedding is available in one language but not in the other, it is initialized as a zero-vector in the second language. Totally unseen words are initialized uniformly at random in the range $[-0.1, 0.1]$. Thus, for every language pair, we obtain word embeddings $\mathbf{w}_L$.

On the character level, we encode every word using a BiLSTM, to which we apply max-pooling to obtain the token-level representation. That is, for a sequence of T characters, $\{c^t\}_{t=1,\dots,T}$ a standard BiLSTM computes two sets of T hidden states, one for each direction. The hidden states are subsequently concatenated for each timestep to obtain the final hidden states, after which a max-pooling operation is applied over their components:

$$\begin{aligned}\overrightarrow{\mathbf{h}_t^c} &= \overrightarrow{\text{LSTM}_t}(\mathbf{c}^1, \dots, \mathbf{c}^t)\\ \overleftarrow{\mathbf{h}_t^c} &= \overleftarrow{\text{LSTM}_t}(\mathbf{c}^t, \dots, \mathbf{c}^T)\\ \mathbf{w}_{char} &= \max(\{[\overrightarrow{\mathbf{h}_t^c}, \overleftarrow{\mathbf{h}_t^c}]\}_{t=1,\dots,T})\end{aligned}$$

We take inspiration from context-attentive embeddings (Kiela et al., 2018), in that we learn weights over the embeddings, but do not include the contextual dependency for reasons of efficiency given the shared task's tight deadline. That is, we combine the language-specific word embeddings $\mathbf{w}_{L_1}$ and $\mathbf{w}_{L_2}$ with the character-level word representation via a simple self-attention mechanism:

$$\begin{aligned}\alpha_i &= \text{softmax}(U \ \tanh(V \ [\mathbf{w}_{L_1}, \mathbf{w}_{L_2}, \mathbf{w}_{char}])),\\ \mathbf{w}_{word+char} &= [\alpha_1 \mathbf{w}_{L_1}, \alpha_2 \mathbf{w}_{L_2}, \alpha_3 \mathbf{w}_{char}]\end{aligned}$$

2.2 Capitalization

Additionally, we concatenate an embedding to indicate the capitalization of the word, which be either no-capitals, starting-with-capitals or all-capitals:

$$\mathbf{w} = [\mathbf{w}_{word+char}, \mathbf{w}_{cap}]$$

This is already captured by the character-level encoder, but made more explicit using this method.

2.3 Shortcut-Stacked Sentence Encoders

The final word representations $\mathbf{w}$ are fed into a stacked BiLSTM with residual connections (i.e., "shortcuts"). This type of architecture has been found to work well for text classification, in conjunction with a final max-pooling operation (Nie and Bansal, 2017). Denoting the input and hidden state of the i-th stacked BiLSTM layer at timestep t as $\mathbf{x}_t^i$ and $\mathbf{h}_t^i$ respectively, we have:

$$\mathbf{x}_t^i = \begin{cases}\mathbf{w}_t & i = 1\\ [\mathbf{w}_t, \mathbf{h}_t^1, \dots, \mathbf{h}_t^{i-1}] & i > 1\end{cases}$$

2.4 CRFs for NER

The hidden states of the last stacked BiLSTM layer are fed into a CRF (Lafferty et al., 2001). CRFs are used to estimate probabilities for entire sequences of tags $\mathbf{s}$ corresponding to sequences of tokens $\mathbf{x}$:

$$\begin{aligned}p(\mathbf{s}|\mathbf{x};\mathbf{w}) &= \frac{\exp(\mathbf{w}\cdot\Phi(\mathbf{x},\mathbf{s}))}{\sum_{\mathbf{s}'}\exp(\mathbf{w}\cdot\Phi(\mathbf{x},\mathbf{s}'))}\\ &= \frac{\exp(\sum_j \mathbf{w}\cdot\phi_j(\mathbf{x}, j, s_{j-1}, s_j))}{\sum_{\mathbf{s}'}\exp(\sum_j \mathbf{w}\cdot\phi_j(\mathbf{x}, j, s'_{j-1}, s'_j))}\\ &= \frac{\prod_j \exp(\psi_j(\mathbf{w}, \mathbf{x}, j, s_{j-1}, s_j))}{\sum_{\mathbf{s}'}\prod_j \exp(\psi_j(\mathbf{w}, \mathbf{x}, j, s'_{j-1}, s'_j))}.\end{aligned}$$

To make the CRF tractable, the potentials must look only at local features. We experiment with two different score functions ψ_j. One that uses bigrams:

[1]Available at https://fasttext.cc/docs/en/crawl-vectors.html.

$$\psi_j(\mathbf{w},\mathbf{x},j,p,q) = \mathbf{W}^{\intercal}_{[p,q,:]}\mathbf{x}_j + \mathbf{B}_{[p,q]},$$

where $\mathbf{W} \in \mathbb{R}^{|S|\times|S|\times H}$ is $\mathbf{w}$ but unflattened, $|S|$ is the number of possible tags, H is the dimensionality of the encoder's features $\mathbf{x}$ and $\mathbf{B} \in \mathbb{R}^{|S|\times|S|}$ is a bias matrix; and a smaller score function with unigrams:

$$\psi_j(\mathbf{w},\mathbf{x},j,p,q) = \mathbf{W}^{\intercal}_{[q,:]}\mathbf{x}_j + \mathbf{B}_{[p,q]}.$$

where instead $\mathbf{W} \in \mathbb{R}^{|S|\times H}$. The terms in the score function can be thought of as the emission and transition potentials, respectively.

3 Implementational Details

3.1 Preprocessing

The noisy nature of the data makes it necessary to apply appropriate preprocessing steps. We apply the following steps to the Twitter data:

- Replaced URLs with [url]
- Replaced users (starting with @) with [user]
- Replaced hashtags (starting with # but not followed by a number) with [hash_tag]
- Replaced punctuation tokens with [punct]
- Replaced integer and real numbers by [num]
- Replaced [num]:[num] with [time]
- Replaced [num]-[num] with [date]
- Replaced emojis[2] by [emoji]

In addition, we found that the Arabic tokenizer may have been imperfect: some words still had punctuation attached to them. In order to mitigate this, we removed any leading and trailing punctuation from tokens for MSA-EGY.

3.2 Training

The LSTMs are initialized orthogonally (Saxe et al., 2013), and the attention mechanism is initialized with Xavier (Glorot and Bengio, 2010). Word embeddings are kept fixed during training, but character embeddings and capitalization embeddings are updated. We set dropout to 0.5 and optimize using Adam (Kingma and Ba, 2014) with a learning rate of $4e^{-4}$ and batch size of 64. We shrink the learning rate with a factor or 0.2 every time there has been no improvement for one epoch, until a minimum learning rate of $1e^{-5}$. We early stop on the validation set, optimizing for F1. We sweep over the two CRF types and BiLSTM hidden dimensions via grid search, trying $[128, 128]$, $[128, 128, 128]$, $[64, 128]$, $[64, 128, 128]$, $[64, 64, 128, 128]$ and $[64, 128, 128, 128]$.

[2]We used the emojis in https://pypi.org/project/emoji/.

Model	Dev F1	Test F1
Baseline	68.17	60.28
Ours	67.74	62.39

Table 1: Results for ENG-SPA.

Model	Dev F1	Test F1
Baseline	79.55	70.08
Ours	81.41	71.62

Table 2: Results for MSA-EGY.

4 Results & Discussion

For both tasks, we compare the proposed model to a simpler baseline where we simply concatenate the FastText embeddings as input to the network.

Table 1 shows the results for ENG-SPA. We observe that our system outperforms the baseline on the test set. The dev set for this task was very small (832, versus a test set of 15.6k), which explains the discrepancy between dev set and test set performance—this discrepancy also made it difficult to tune hyperparameters properly for this task. We also tried a very simple ensembling strategy, where we took our top three models and randomly sampled a response, which only marginally improved test score performance to 62.67. We did not pursue proper ensembling due to time constraints. The best performing model had hidden dimensions $[128, 128, 128]$ and used the bigram CRF.

The results for the MSA-EGY task are reported in Table 2. While English and Spanish are two distinct languages, Modern Standard Arabic and Egyptian are more closely related, leading to interesting challenges. We observe a similar improvement in this task. As noted in the previous section, we did find that this task required slightly different preprocessing. We did not try any ensembling strategies on this task. The best performing model

	Precision	Recall	Entity F1
EVENT	56.25	20.00	29.51
GROUP	69.77	30.93	42.86
LOC	70.75	69.23	69.98
ORG	62.50	27.23	37.93
OTHER	14.29	1.71	3.08
PER	76.52	68.15	72.09
PROD	63.76	47.53	54.46
TIME	51.58	37.09	43.24
TITLE	49.14	25.79	33.83
Overall	70.62	55.88	62.39

Table 3: ENG-SPA test performance breakdown.

	Precision	Recall	Entity F1
EVENT	78.18	61.43	68.80
GROUP	69.77	76.92	73.17
LOC	76.19	67.84	71.78
ORG	66.14	67.20	66.67
OTHER	100.00	100.00	100.00
PER	77.29	69.53	73.21
PROD	76.47	78.79	77.61
TIME	64.29	72.00	67.92
TITLE	31.58	60.00	41.38
Overall	73.95	69.42	71.62

Table 4: MSA-EGY test performance breakdown.

had hidden dimensions $[64, 64, 128, 128]$ and used the unigram CRF.

While developing our system, we made some interesting observations. For instance, we noticed that performance on the Event and Time categories was greatly improved through preprocessing the numbers and splitting out patterns into date and time categories. Adding explicit capitalization features improved performance on the Person, Location and Organization categories. Tables 3 and 4 show a breakdown of the performance per task by category on the respective test sets. It is interesting to observe that the Title category is consistently hard for both tasks. The Other category was perfectly handled for MSA-EGY, while this was very bad for ENG-SPA — this could however also be an artifact, since that category was quite small.

We felt that we could have benefited from having a strong gazetteer, but also believe that this would kind of defeat the purpose of our general neural network architecture, which should not have to rely on those kinds of features.

5 Conclusion

Dealing with code-switching is a prominent problem in handling noisy user-generated social media data. The tendency for speakers to code-switch poses difficulties for standard NLP pipelines. Here, we described our work on the shared task: we introduced a system that performs self-attention over pre-trained or character-encoded word embeddings together with a shortcut-stacked sentence encoder. The system performed impressively on the task. In the future, we would like to analyze the system to see whether it has indeed learned to "code-switch" via embedding attention.

References

Gustavo Aguilar, Fahad AlGhamdi, Victor Soto, Mona Diab, Julia Hirschberg, and Thamar Solorio. 2018. Overview of the CALCS 2018 Shared Task: Named Entity Recognition on Code-switched Data. In *Proceedings of the Third Workshop on Computational Approaches to Linguistic Code-Switching*, Melbourne, Australia. Association for Computational Linguistics.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of LREC*.

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

Douwe Kiela, Changhan Wang, and Kyunghyun Cho. 2018. Context-attentive embeddings for improved sentence representations. *arxiv preprint arXiv:1804.07983*.

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

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

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

Xuezhe Ma and Eduard H. Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. pages 1064—1074.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Learning word vectors for 157 languages. In *Proceedings of LREC*.

Yixin Nie and Mohit Bansal. 2017. Shortcut-stacked sentence encoders for multi-domain inference. *arXiv preprint arXiv:1708.02312*.

Younes Samih, Suraj Maharjan, Mohammed Attia, Laura Kallmeyer, and Thamar Solorio. 2016. Multilingual code-switching identification via lstm recurrent neural networks. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 50–59.

Andrew M Saxe, James L McClelland, and Surya Ganguli. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.