# CS5740 Assignment 5 Github Group: technologicaa

**Chaoping Deng**
cd446@cornell.edu

**Yating Zhan**
yz2237@cornell.edu

## 1 Introduction

In this project, we built a Named Entity Recognition tagger for Twitter text. We used a bidirectional LSTM and conditional random field to build a model for this task. In addition to provided training data, we used GloVe word embedding to achieve a better F1 score. Eventually, our F1 score on test set is 0.5720.

In section 2, we will explain the models that we used, in section 3 and 4, we will talk about experiments and results, and in section 5, we will discuss error analysis and conclusion.

## 2 Approaches

### 2.1 CRFs

The main sequence to sequence inference model we used is Conditional Random Fields. It captures the tagging information of the whole sequence rather than only considering the previous tag information. Our first approach is to use hand craft features for CRFs. We used a CRFs model from sklearn crfsuit to update weights . CRFs used

$$\frac{\delta}{\delta w_j}L(w) = \sum_{I=1}^{m}(\phi(X^i, Y^i) - \sum_{Y} p(Y|X^i; w)\phi_j(X^i, Y)) - \lambda w_j))$$

to update the weights. Because the this update will require a time complexity of number of label to the power of number of words in a sequence and it's hard realize, the CRF model in sklearn use forward and backward to approximate the weight update. To decode the encoded sequence to tags, we used viterbi algorithm to maximize norm in

$$norm(i, y_i) = \sum_{y_i - 1} exp(w \cdot \phi(X, i, y_{i-1}, y_i)) norm(i-1, y_{i-1})$$

for decoding. We used (Robert, 2016) for the viterbi code.

### 2.2 LSTMs

Because hand craft features has its limitations, we used a bidirectional LSTM to encode sentences and use the sentence encoding as features for CRF. LSTMs helps the model to remember contextual information by using output gate($o_t$), forget gate ($f_t$), and input gate($i_t$). For each input $x_t$ and hidden state $h_{t-1}$, LSTMs compute the gates so that the model can learn which words it should remember for longer and which words it should forget. The gates are computed using the following formulas:

$$o_t = \sigma(W^o[h_{t-1}, X_t] + b^o)$$

$$f_t = \sigma(W^f[h_{t-1}, X_t] + b^f)$$

$$i_t = \sigma(W^i[h_{t-1}, X_t] + b^i)$$

We used the Dynet LSTMBuilder in our model for encoder and decoder. We followed (Guillaume Lample, 2016) to build a BiLSTM network to encode tokens in a sentence.

### 2.3 Word Embedding

- Pre-trained word embedding
  For word-level word representations, we used the pre-trained word embeddings from GloVe(Pennington et al., 2014). We looked for the embeddings of each word in the Glove file and retrieved its 200-dimension vector representation and initialized our biLSTM network using these parameters. For words that did not appear in the Glove dictionary, we simply initialize their embeddings using that of the 'Unknown' token. We have also attempted random initialization or zeroing out all unknown vectors. It turns out that the Unknownn' token worked the best
- Character-based word embedding
  To better capture every word's prefix and suffix, and to better understand unknown words, we also used a simple multi-layer perceptron to encode each character features in a word. We randomly initialized character embedding for all the characters in the whole dataset. For each word in a sentence, we pad the word to the maximum length of word to the tail, then column-wise concatenate the embedding of the characters to form a matrix C of [dimension character embedding size $\times$ maximum word length]. We then use $C \times W_c + b_c$ to get the word embedding.
- Orthographic character-based word embedding
  In (Limsopatham and Collier, 2016),the author argues that using ordinary character embedding has the risk of introducing much noise into the system.In order to capture the shape information regarding characters. We would also want to categorize each word to special representations according to the following convention:for each character token in a word, convert the character to 'p' if the it is not a digit or letter, otherwise convert it to 'n' if it is a digit, otherwise if convert it to 'C' and 'c' if it is upper/lower case respectively. Therefore, we would convert the following sentence "@abc123 I love Nlp" to "pcccnnn C cccc Ccc". This essentially covers the structural information of a related word through character-level embedding

After we get pre-trained word embedding, the character-based embedding, and the orthographic character-based word embedding, we concatenate all word embeddings together as the input layer for the bidirectional LSTMs.

### 2.4 Part-Of-Speech Embedding

We used a Twitter Part-of-Speech tagger trained by (Olutobi Owoputi, 2012) to capture the POS tag for each word. The POS tagger inference a POS tag for each word and provide confidence level for each inference. We only consider the tags with confidence level higher than 0.8 and we used 'UNK' for the tags have confidence level lower than 0.8. We randomly initialized POS embedding for all the POS tags and concatenate a given word's POS tag embedding with it's word embeddings for the LSTMs.

### 2.5 Final Model

Figure 1 shows the final whole model architecture.

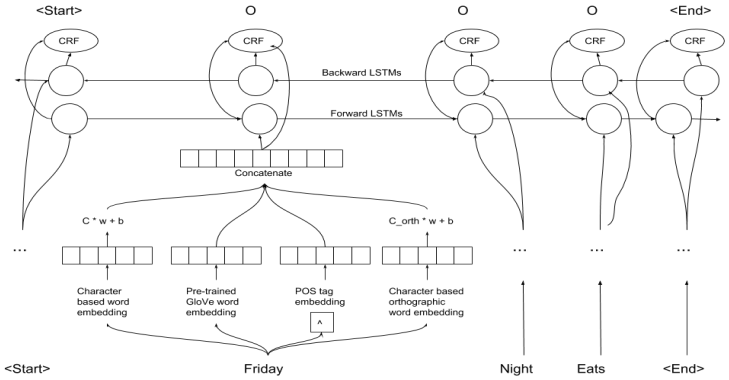


Figure 1: Full model architecture

## 3 Experimental Setup

### 3.1 Data Overview

We are using provided Twitter data with Named Entity Recognition labels in training and development set. There are 2394 training sentences and 959 development set sentences.

### 3.2 Preprocessing

There are two major characteristics in the dataset that makes the vocabulary large and messy. First, the sentences in the data set often contain words starting with "@" and it representing tagging another Twitter account. But the string of another Twitter account often is unique and out of context. Second, a lot of the sentences end contain an unique url. To make the vocabulary cleaner, we did the follow mutation for the each sentence in training, development, and test data set.

- Change all words starting with "@" to just "@".
- Change all words starting with "http" to just "http".

After preprocessing, we have 16061 unique words and 484 unique characters. For orthographic character representation, we have 1072 unique words and 4 unique characters (C,c,p,n). The total of number of unknown words that are not contained in the glove word embedding file is

### 3.3 Feature Generation

#### 3.3.1 Hand Craft Features

We experimented with using hand craft features for CRFs. There are two main categories of features that we used:

- word feature: prefix of current word, suffix of current word, if word is digit, if contains capitalized letters.
- context: the previous word and its word feature, the next word and its word feature.

#### 3.3.2 Features trained by LSTMs

For each word in a sentence, we concatenate word embedding mentioned in section 2.3 and POS embedding mentioned in section 2.4 to form a input layer for the LSTMs layers. We then used the output layer of LSTMs as the features for CRFs.

### 3.4 Stop Criteria

We stop the training by identifying saturation for loss score of the model. Specifically, for every epoch, if the loss doesn't decrease for more than 0.0001, we stop the training process.

| Features | F1 on Dev | Time (sec/epoch) |
|---|---|---|
| CRF + hand craft features | 0.4615 | 32(total) |
| biLSTM + CRF | 0.2123 | 13 |
| Pre-train word + biLSTM + CRF | 0.4882 | 14 |
| Character_embedding + Pre-train word+ biLSTM + CRF | 0.5635 | 19 |
| Character Orthographic embedding + Character _embedding + pre-train word + biLSTM +CRF | 0.6092 | 23 |
| POS_Tagger +Character Orthographic embedding + Character_embedding + pre-train word + biLSTM +CRF | 0.6186 | 25 |

Table 1: Feature Ablation

## 4 Results and Analysis

### 4.1 Ablation

We have compared 6 different sets of model and parameter combination. The worst performing model utilizes handcraft features that we carefully selected. By comparison, A single-layer bi-directional LSTM without word embedding does not really capture the association between different words. Specifically, there are 7000 unknown words which appears in the development set but not the training set. In order to capture information relating to these words, we need external pre-trained embedding to provide such information relating to word meanings. We saw significant increase in F1 value with addition of pre-trained twitter word vectors. The addition of character-level embedding of words further captures information for noisy user generated texts. Specifically, twitter texts contains many words and sentences with broken structures or wrong spellings.Our hypothesis is that this has made character-level embedding more meaningful than word level embedding; After adding them, we indeed observed an significant increase of performance from 0.48 to 0.56. Next, real-world texts are extremely complicated and even the character-level word embeddings contain lots of noise. We would like to use the orthographic information of different characters to train the network to understand different word structures, specifically for formal words, terms and slangs. Lastly,we think that adding POS tag information would add an additional reference regarding syntactical information of words.

### 4.2 Hyper Parameters

We experimented with some hyper parameters for a better result of our model. The hyper parameters include:

- LSTMs hidden layer dimension: 100, 150, 200

- Character based embedding dimension: 50, 100
- POS embedding dimension: 50, 100

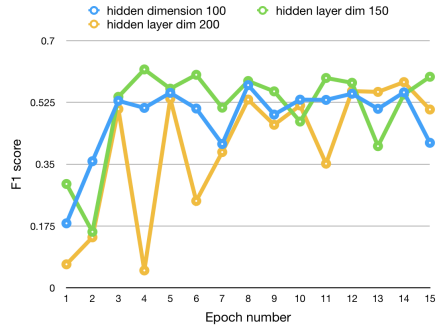


Figure 2: LSTM hidden layer dimensions plots with character based embedding dimension 50, POS embedding dimension 50

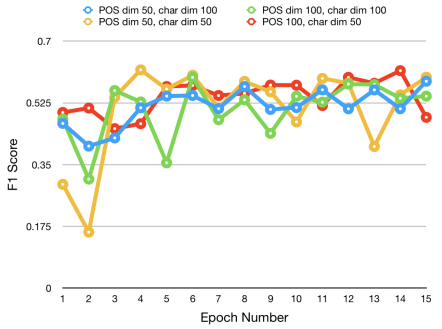


Figure 3: Character based embedding dimension and POS embedding dimension plots with LSTMs hidden layer dimension 150.

### 4.3 Final Model Performance

Our submitted result achieved 0.6091 on the development dataset and 0.5720 on the test set. However, this is not our best performed data. As shown on the Figures 2 and 3, our best model achieved 0.6186 on the development set with hidden layer dimension 150, character based embedding dimension 50, POS tag dimension 50 for 4 epochs. The runtime is about 2 minutes.

## 5 Error Analysis and Conclusion

### 5.1 Error Analysis

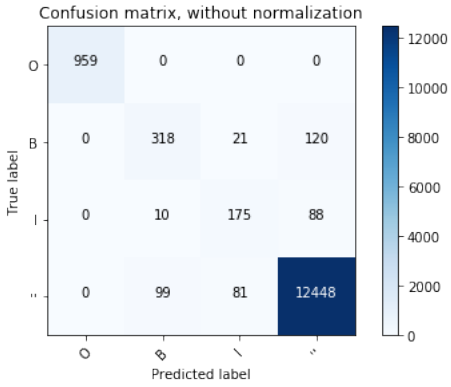


Figure 4: Confusion matrix

As shown in Figure, We expected the errors between B,I with O. We identify the following major cases where the model makes errors:

- Miss-classified slangs: The words which are supposed to be 'O', the model predicts them to be I or B because there are internet slangs that appear to be 'Special'. Such examples include: ['#StoryOfMyLifeFollow-Party','#PitBullsandParolees','News','Descarga','TGFbro', 'RT',...]
- Regular words used in special situations: such words include cases where an entity is wrongly classified as non-entities.Such examples include:['The','dance','And','Without', 'Words','Without','IN','PhotoS','Podcast','Web',...]. These words, if used independently, are indeed non-entites. However, for cases such as, The "TC Gadgets Podcast" or "the album 'DifferenT PhotoS', they are used as part of an entity.

| | Precision | Recall | F1 | Support |
|---|---|---|---|---|
| O | 1.00 | 1 | 1 | 959 |
| B | 0.74 | 0.69 | 0.72 | 459 |
| I | 0.63 | 0.64 | 0.64 | 273 |

Table 2: Model Performance

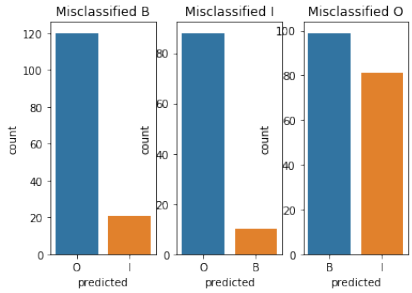


Figure 5: Classifications

### 5.2 Conclusion and Improvement

In this paper we have experimented with multiple models and compared their performances on the Named-Entity Recognition Task.With time allowance, we can further improve the performance of the model by using: 1.Encode tags from 'B','I','O' to 'I' and 'O' and post process the predictions to convert the first 'I' tags to 'B'. This essentially simplify the task. 2.Add word-level orthographic embed-dings. We can add word-level orthographic embeddings in the way we added word-vectors.The difference is that we construct a look-up table for transformed words such as 'Ccccn' etc 3.Add a more sophisticated embedding network. Instead of simply feeding word and character level embed dings through a hidden layer ,we can leverage RNN or CNN to create embedding which better captures word features

## References


Miguel Ballesteros Sandeep Subramanian Kazuya Kawakami Chris Dyer Guillaume Lample. 2016. Neural architectures for named entity recognition.

Nut Limsopatham and Nigel Collier. 2016. Bidirectional lstm for named entity recognition in twitter messages. In *NUT@COLING*.

Brendan O'Connor Chris Dyer Kevin Gimpel Nathan Schneider Olutobi Owoputi. 2012. Cmu ark twitter part-of-speech tagger. https://github.com/brendano/ark-tweet-nlp.

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

Robert. 2016. Bilstm-crf for sequence labeling in dynet. https://github.com/rguthrie3/BiLSTM-CRF.