

# Neural Network Model for NER

Juan Wang, Yukai Wu

## Abstract

In this project, I apply a variety of neural sequence models to achieve NLP sequence tagging. These models include BiLSTM-CRF network, sequence-to-sequence (seq2seq) BiLSTM-CRF network, Seq2Seq-BiLSTM-Attention-CRF network and Seq+Seq-BiLSTM-CRF network. I will show that BiLSTM-CRF model can efficiently use both past and future input features thanks to a bidirectional LSTM component. It can also use sentence level tag information thanks to CRF layer. BiLSTM-CRF model can produce a relatively good accuracy on NER data sets. Thus, I use this model as my baseline for the experiments of other three models. Also I give the analysis and comparisons among these models. After experiments, the last Seq+Seq-BiLSTM-CRF model is the winner which can produce state of the art accuracy on the NER task.

## Introduction

Information Extraction (IE), one of the important tasks in text analysis and Natural Language Processing (NLP), involves extracting meaningful pieces of knowledge from unstructured information sources (as unstructured data is computationally opaque), and automatically transforming into structured representation that can be interpreted and manipulated by machines. Named Entity Recognition (NER), a term first coined at the sixth Message Understanding Conference (MUC6), is a sub-task of IE, the goals of which are identifying references of named entities in unstructured documents and classifying them into fine-grained ontological concepts. NER plays a vital role in reference resolution, other types of disambiguation, and meaning representation in other natural language processing applications. The process depends on a number of fundamental Natural Language Processing (NLP) steps such as tokenization, part-of-speech tagging, parsing and model building. Moreover, NER is also an important technology for many applications and research areas.

- Classifying Content for News Providers

NER can automatically scan entire news articles and reveal which are the major people, organizations, and places discussed in them. Knowing the relevant tags for each article helps in automatically categorizing articles in defined hierarchies and enables smooth content discovery. For example, after entering a text “When Michael Jordan was at the peak of his powers as an NBA superstar, his Chicago Bulls teams were mowing down the competition, winning six National Basketball Association titles”, it will get the result of some keywords like “Place: Chicago”, “Name: Michael Jordan” and “Group: National Basketball Association”.

- Powering Content Recommendations

For news publishers, using NER to recommend articles according to individuals’ addictive-like reading behaviors can make their platforms more engaging. This can be done by extracting entities from a particular article and recommending the other articles which have the most similar entities mentioned in them. This is an approach that can be also applied to develop content recommendations for a media industry client.

- Information Retrieval (IR) and Question Answering (QA)

Current text-based question answering (QA) systems usually contain a named entity recognizer (NER) as a core component. The rationale of incorporating a NER as a module in a QA system is that many fact-based answers to questions are entities that can be detected by a NER. Named entities are identified to locate important information and

facts (Lee et al., 2007; Srihari and Peterson, 2008). For example, in the question answering competition in TREC-8 (TREC-8 QA Data, 2002), 80% of the evaluation questions ask for a named entity (Nadeau, 2007). Therefore, by incorporating in the QA system a NER, the task of finding some of the answers is simplified considerably.

- Semantic Web

NER is used to improve semantic search (Caputo et al., 2009). An IR system, called SENSE (SEmantic N-levels Search Engine), manages documents indexed at multiple separate levels: keywords, senses (word meanings) and entities (named entities). The system is able to combine keyword search with semantic information provided by the two other indexing levels.

- Machine Translation

Accurate translation of named entities plays an important role in the translation of the overall text (Babych and Hartley, 2003). By using the GATE NE recognition module as a preprocessor for commercial state-of-the-art MT systems, the high-quality automatic NE recognition could be used to create do-not-translate (DNT) lists of organization names, a specific type of NE which in human translation practice is often left untranslated.

- Customer Support

There are a number of ways to make the process of customer feedback handling smooth and Named Entity Recognition could be one of them. If you are handling the customer support department of an electronic store with multiple branches worldwide, you go through a number mentions in your customers' feedback. This can be then used to categorize the complaint and assign it to the relevant department within the organization that should be handling this. Similarly, there can be other feedback tweets and you can categorize them all on the basis of their locations and the products mentioned. You can create a database of the feedback categorized into different departments and run analytics to assess the power of each of these departments.

## Project Goals

My main objective is to apply a model result in producing desirable accuracy on the topic task. In those relative research papers, some of them demonstrated that their models obtain state-of-the-art performance in NER and the highest F1 scores can be up to 90.94. Another paper which performed experiments on two tasks, NER and CCG supertagging, observed improvements from their approach on both the tasks, even though seq2seq model are more appropriate for CCG supertagging rather than NER. Thus, I am motivated by them and try to build my own neural sequence model to do the experiment. Hopefully the model can get the same accuracy or even higher. I also want to make the comparison between these model performances on NER. Further, I will give an in-depth analysis of the impact of these neural network components on each model. At last, I will use the best performance model to run an example from (Ratinov, 2009):

*LONDON 1996-12-06 Dutch forward Reggie Binker had his indefinite suspension lifted by FIFA on Friday and was set to make his Sheffield Wednesday comeback against Liverpool on Saturday. Binker missed his club's last two games after FIFA slapped a worldwide ban on him for appearing to sign contracts for both Wednesday and Udinese while he was playing for Feyenoord.*

By comparing with the correct tagging like the output below, we can see how successful this model can be:

*[LOC LONDON] 1996-12-06 [MISC Dutch] forward [PER Reggie Binker] had his indefinite suspension lifted by [ORG FIFA] on Friday and was set to make his [ORG Sheffield Wednesday] comeback against [ORG Liverpool] on Saturday. [PER Binker] missed his club's last two games after [ORG FIFA] slapped a worldwide ban on him for appearing to sign contracts for both [ORG Wednesday] and [ORG Udinese] while he was playing for [ORG*

Feyenoord].

## Motivation

In this section, I will give the motivation of the fundamental model design including analysis of the advantages and disadvantages of each model.

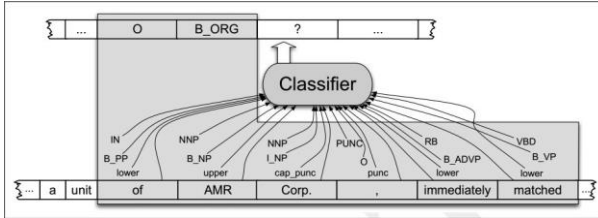


Figure 1: Traditional NER: The features available to the classifier during training and classification are those in the boxed area.

Figure 1 illustrates the operation of this traditional sequence labeler at the point where the token Corp. is next to be labeled. We can see there is a context window that includes the 2 preceding and following words, then the features available to the classifier are those shown in the boxed area. The more powerful models are, the larger the window size become. As a result, the memory requirements of the system grow exponentially. Thus, it nearly impossible to model large word windows without running out of memory.

Unlike the conventional NER models, where only a finite window of words would be considered for conditioning the model, Recurrent Neural Networks (RNN) are capable of conditioning the model on all previous words, meaning that the state at time  $t$  captures information from the past,  $x_0, \dots, x_{t-1}$ , and the present input  $x_t$  (see figure 2).

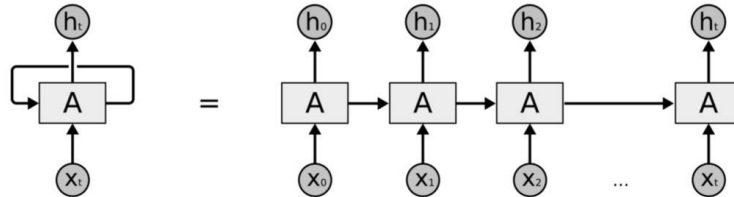


Figure 2: an unrolled recurrent neural network

Given an input sequence  $X_1^T$ , a RNN produces an output sequence  $Y_1^T$ , computed as (see figure 3):

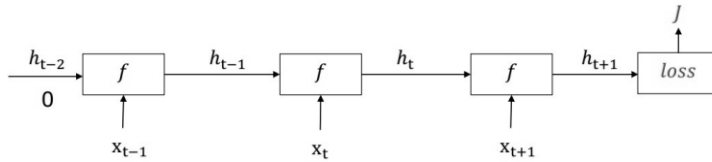


Figure 3: given input sequence, compute the hidden state and loss function.

$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_{t|})$$

$$\hat{y}_t = \text{softmax}(W^{(sy)}h_t)$$

$$J^{(i)}(\theta) = -\sum_{j=1}^{|Y|} y_{t,j} \times \log(\hat{y}_{t,j})$$

Note:  $\sigma$  is an element-wise function (usually the tanh or sigmoid).

Where  $h_t$  is the relationship to compute the hidden layer output features at each time-step  $t$ ,  $y_t$  is the output probability distribution over the vocabulary at each time-step  $t$ . The network is typically trained via stochastic gradient descent (SGD), using the backpropagation to minimize a loss function under some optimality criterion, usually cross-entropy between the output and the training data probability distribution.

Comparing regular RNNs and conventional NER model based on context window, we can find the drawback of the regular RNN, the input sequence of which is only scanned in one direction, normally from past to future. While the context window can include either preceding or following words. Moreover, I want to output a prediction of  $y_t$  that may depend on the whole input sequence. In order to capture both past and future context, bi-directional RNN (BRNN) was necessarily implemented in my project. The main idea is the forward pass abstracts and summarizes the context in the forward direction while the backward pass does the same from the reverse direction. Figure 4 shows a bi-

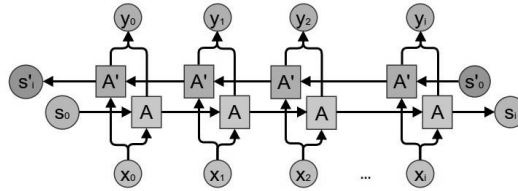


Figure 4: Bi-directional RNN

directional network architecture; at each time-step  $t$ , this network maintains two hidden layers, one for the left-to-right propagation and another for the right-to-left propagation. The final classification result  $y_t$ , is generated through combining the score results produced by both RNN hidden layers, as the equation shown below:

$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

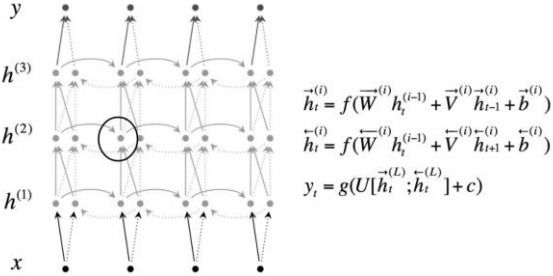
$$\hat{y}_t = g(Uh_t + c) = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$

$h = [\vec{h}; \overleftarrow{h}]$ : represents (summarizes) the past and future around a single token

Further, in my project, a two-layer bi-directional RNN has been achieved where the lower layer can feed the upper layer. In this architecture, at time-step  $t$  each intermediate neuron receives three sets of parameters, one from the previous time-step (in the same RNN layer), two from the previous RNN hidden layer which can separately come from the left-to-right RNN and right-to-left RNN.

The equation can be modified as (see figure 5):

Unfortunately, this vanilla BRNN model still has some problems should address. Recall the goal of a RNN implementation is to enable propagating context information through faraway time-steps. Thus, the basic problem is that gradients propagated over many stages tend to either vanish or explode. Let's consider the previous equation, the partial derivative of  $J$  with respect to  $W_t$  can be computed through applying the chain rule differentiation, and we get the following equation:

$$\frac{\partial J}{\partial W_t} = \frac{\partial J}{\partial h_{t+n}} \frac{\partial h_{t+n}}{\partial h_{t+n-1}} \dots \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial W_t}$$


$$\vec{h}_t^{(i)} = f(\vec{W}^{(i)} \vec{h}_t^{(i-1)} + \vec{V}^{(i)} \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} \overleftarrow{h}_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\vec{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Figure 5: a two-layer bi-directional RNN

In the scalar case, imagine multiplying weight  $W$  by itself many times, the product can become a very large or small number depending on the magnitude of  $W$ . In general case,  $W$  can be seen as the largest eigenvalue of the weight matrix. When  $W$  is larger than 1 and  $n$  is sufficiently large, then the gradient value grows extremely large. In the geometrical perspective, the loss function is like a valley with a steep wall on its one side, our goal is to reach the minimum point by iterative implementing SGD algorithm. Consequently, when stochastic gradient descent hits the wall and does a gradient descent step (which will be a huge step), the gradient is forced to jump across the valley

$$\nabla = \begin{cases} \frac{c}{|\nabla|} \nabla, & \text{if } |\nabla| > c; \\ \nabla, & \text{otherwise} \end{cases}$$

moving perpendicular to the steep walls and pushed off to a faraway location, possibly leaving the valley and disrupting the further learning process. This issue, called the Gradient Explosion problem, can be resolved through a gradient norm clipping strategy. In my project, I also use this strategy by setting a hyperparameter  $c$  as the threshold, see the equation:

That is, whenever the gradients norm reaches a certain threshold, they are rescaled to a fixed size, pulling back the error gradient in a smoother region near the wall, where is close to the original gradient landscape that the SGD can be free to explore other descent directions. Theoretically, setting this threshold need to look at statistics on the average norm over a sufficiently large number of updates. In my practical experiments I noticed that training is not very sensitive to this hyperparameter and the algorithm behaves well even for rather small threshold.

After solving the Gradient Explosion problem, let's move on to handle the opposite situation. When  $W$  is much smaller than 1, the gradient value goes to zero. Similarly, from a geometrical view, it likes going through a flatland, lingering

at that area without proceeding the gradient descent to the bottom of the valley. This issue is called the Gradient Vanishing problem, which can drastically reduce the quality of the model to learn correlation between temporally distance events. According to the previous equation, not only the one factor about the weight  $W$  affects the magnitude of gradients, but also the other factor concerning the activation functions (more precisely, their derivatives) that the gradient passes through. Recall the basic RNN architecture, the unit of which consist of linear functions followed by a nonlinear activation function, such as tanh or sigmoid. As figure 6 shown, each time step computes a function fairly close to the identity function  $f(x) = x$ , but after several steps, it will become a step function which is the cause of the gradient vanishing and explosion. Due to the iterated functions can have such complex and chaotic behavior, one solution of this problem it to use the Rectified Linear Units (ReLU) instead of the tanh or sigmoid function.

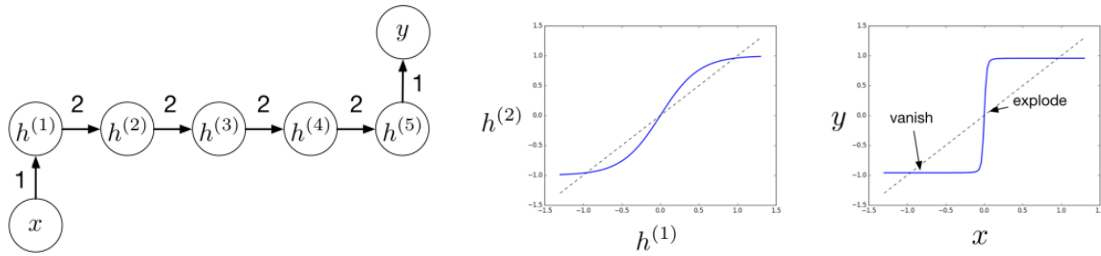


Figure 6 the function computed at each time step & the function computed by the network as a whole

The derivative for the ReLU is either 0 or 1. After clipping the activations to be nonnegative, gradients would flow through the neurons whose derivative is 1 without getting attenuated while propagating back through time-steps. In order to avoid the gradient vanishing and explosion problem, I used the Long-Term Short-Term Memory (LSTM) network with memory blocks in the hidden layer instead of the ordinary RNN with one fully recurrent hidden layer. The LSTM architecture was designed to allow the network to remember information (such as evidence for a particular feature or category) over a long duration. The key to LSTM model is the cell state, the horizontal line running through the top of the diagram (see the figure 7), which makes it have an internal recurrence (a self-loop), in addition to the

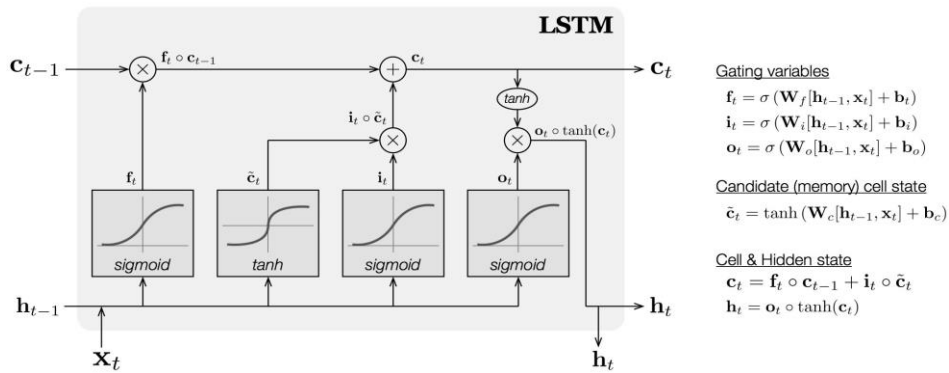


Figure 7: LSTM cell

outer recurrence of the RNN. Recall the aforementioned solution for the notorious problem of vanishing/explosion gradients is by using linear function instead of non-linear function. From the equation  $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$  we can see that, rather than multiplying the cell states, there is a directly linear connection between  $c_t$  and  $c_{t-1}$ , like the ResNet applying in computer vision to handle the degradation problem by introducing the deep residual learning

framework. Besides, another crucial addition, adaptive “forget gates”, has been made to control the weight on the self-loop, by learning to reset memory blocks once the contents are outdated or useless. It not only means immediate resets to zero but also gradual resets corresponding to slowly fading cell state, so the time scale of integration can be changed dynamically. The block also receives input feature which is computed with a regular artificial neuron unit (a tanh activation function squashes the value to between -1 and 1). The value can be accumulated into the memory cell if the sigmoidal input gate allows it. Likewise, the output from the blocks, which is the value of the memory cell passed through a tanh activation function, can be shut off by the modulation of an output gate (the same form as forget and input gates).

Therefore, the LSTM’s ability to learn long-term dependencies more easily than the simple recurrent architectures makes it a rational choice for this project.

## Data

I used the CoNLL-2003 named entity data as my datasets which consists of files covering two languages: English and German. But I only choose the English language including a training file, a development file, and a test file. The current model was trained with the training data. Based on the result of the comparison between output from the specific learning algorithm and gold output denoted as target, the parameters of the model are adjusted. When it reached the checkpoint, the development data could provide an evaluation of the model fit on training dataset while tuning the model’s hyperparameters. Also, it can be used for regularization by setting a patience as the stopping sign, which can stop training when this flag out of patience. Finally, when the best parameters were found, the model could be tested on the test data which provides an unbiased evaluation of a final model. The results of the different learning methods on the test sets were compared in the evaluation of the task. The split between development data and test data was chosen to avoid systems being tuned to the test data.

Here is the training procedure:

```
def train()
    prepare training data and dev data
    build model
    while epoch
        train random batch-size sentences
        if (epoch%3==0)
            evaluate dev data
    end
    save model
```

The English data was taken from the Reuters Corpus2. This corpus consists of Reuters news stories between August 1996 and August 1997. For the training and development set, ten days’ worth of data were taken from the files

English data	Articles	Sentences	Tokens	English data	LOC	MISC	ORG	PER
Training set	946	14,987	203,621	Training set	7140	3438	6321	6600
Development set	216	3,466	51,362	Development set	1837	922	1341	1842
Test set	231	3,684	46,435	Test set	1668	702	1661	1617

Figure 8: the statistical data of the English data set

representing the end of August 1996. For the test set, the texts were from December 1996. The given dataset included the corpus after some linguistic preprocessing had been done: a tokenizer, part-of-speech tagger, and a chunker were applied to the raw data. The English data was tagged and chunked by the memory-based MBT tagger (Daelemans et al., 2002). Named entity tagging of English training, development, and test data, was done by hand at the University of Antwerp. Mostly, MUC conventions were followed (Chinchor et al., 1999). An extra named entity category called MISC was added to denote all names which are not already in the other categories. This includes adjectives, like Italian, and events, like 1000 Lakes Rally, making it a very diverse category. (see figure 8)

As for the data format, all data files contain one word per line with empty lines representing sentence boundaries. At the end of each line there is a tag which states whether the current word is inside a named entity or not. The tag also encodes the type of named entity. Here is an example sentence (figure 9):

U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

Figure 9: the data format each line includes the raw data, POS tagger, chunk tagger, named entity tagger

## Experiment 1(baseline): Bi-LSTM-CRF

This is my first model which combine the bi-directional LSTM network with a CRF network on the top layer. (see figure 10)

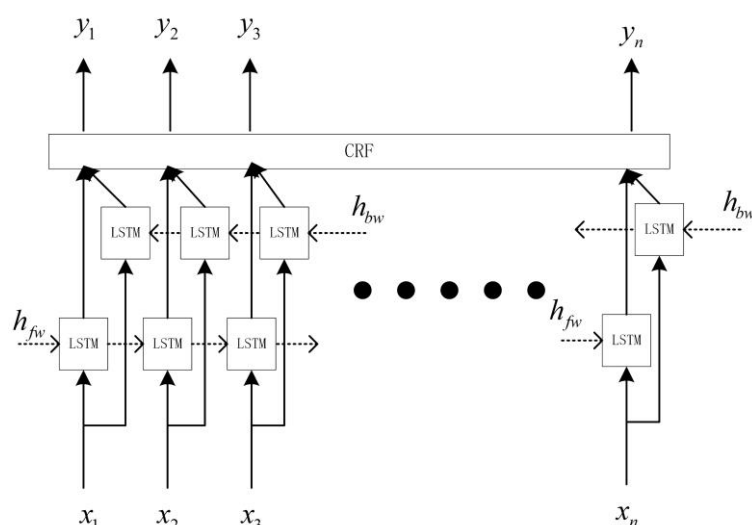


Figure 10: Bi-directional LSTM-CRF

## Word Embeddings

Almost all Deep Learning Architectures are incapable of processing strings or plain text in their raw form. In order to



make a model understand and process the natural language, we need to transform the free-text words into numeric values. If we do a one-hot encoding in which each distinct word stands for one dimension of the resulting vector and a binary value indicates whether the word presents (1) or not (0), this approach of representation will demand hundreds of thousands of dimensions (recall our entire vocabulary size for corpus is 20104). Thus, one-hot encoding is impractical computationally when dealing with this situation. Instead, I used word embedding to represent words in vectors of numeric values with much lower and denser dimension.

- Every word in a sentence will be transform into a digit according to the order number in the corpus vocabulary of size  $V_c$ . (See figure 11)  
(For example, sentence “EU rejects German call to boycott British lamb.”, will be represented as “932 18956 224 734 11 4345 202 6261 4”)

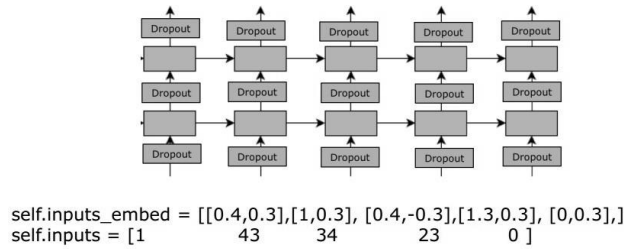


Figure 11: input word embedding

- Searching every digit  $i_t$  in the input word embedding matrix  $W$  of size  $V_c \times d$ , by getting the  $i$ -th row of the matrix  $W$ , we will get the  $d$ -dimension embedded vector, which is the input of LSTM.
- The output of LSTM is also a  $d$ -dimension vector. Likewise, after output word embedding matrix  $W'$  of size  $V_l \times d$ , we will get output one-hot encoded vector  $h_t$  of size  $V_l$  which is the size of target label set.

Note: despite the name,  $W'$  is independent of  $W$ , not a transpose or inverse.

### Beam-search

Previous model, by taking input sequence  $(i_1, i_2, \dots, i_T)$ , will return another sequence  $(h_1, h_2, \dots, h_T)$ . Due to using bi-directional LSTM, the representation of a word is obtained by concatenating its left and right context representations,  $h_t = [\vec{h}_t; \overleftarrow{h}_t]$ . A suboptimal tagging model is to predict a distribution of tags for each time step and then use beam search to find optimal tag sequences, like using maximum entropy classifier and Maximum entropy Markov models (MEMMs). Relative to simpler greedy methods, beam search is a desirable choice of test-time decoding algorithm. For the algorithm, the computational complexity is  $O(\text{Blog}(V)VN)$ , while spatial complexity is  $O(BN)$ .

However, the typical approach to training RNN models is to use a locally normalized maximum likelihood objective (cross-entropy training). Such procedure is not search-aware, that means it does not directly consider the behavior of the final model. As a result, for cross-entropy trained models, beam decoding can sometimes yield reduced test performance when compared with greedy decoding. Currently, there are two approaches to solve this problem.

One is to form a sub-differentiable surrogate objective by introducing a novel continuous approximation of the beam search decoding procedure. It is much like reinforcement learning or imitation learning which forgoes direct optimization of a global training objective, instead incorporating credit assignment for search errors by using methods like early updates that explicitly track the reachability of the gold target sequence during the search procedure. By directly optimizing a continuous and global training objective using backpropagation, as a result, credit assignment is

handled directly via gradient optimization in an end-to-end computation graph.

## CRF

Beam search can find an approximate optimal solution but can't guarantee acquiring global optimum. Besides, NER is one such task that have strong dependencies across labels. Therefore, the second approach is leading to Conditional

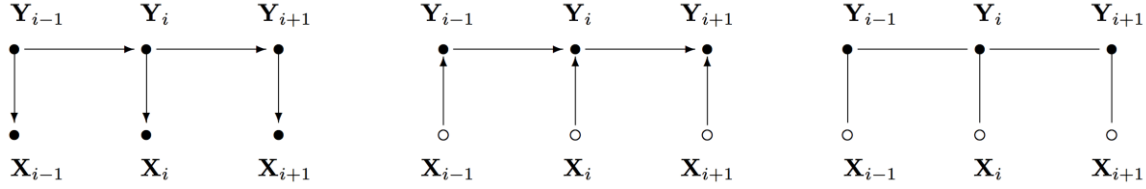


Figure 12: graphical representation for HMM, MEMM, and CRF (which is undirected graph)

Random Fields (CRF) models to focus on sentence level instead of individual position. The figure 12 below shows the graphical representation for the Hidden Markov Model, the Maximum Entropy Markov Model and the Conditional Random Fields.

They are all sequence prediction model, the critical difference between CRF and MEMM is that the latter compute the probability of the next state, given the current state and the observation, whereas CRF computes all state transitions globally in a single model.

In CRF, an undirected graphical model is:

$$p(s_1 \dots s_m | x_1 \dots x_m) = p(\underline{s} | \underline{x})$$

Note:  $\underline{x}$  refers to an input sequence  $x_1 \dots x_m$ , and  $\underline{s}$  refers to a sequence of states  $s_1 \dots s_m$ .

Then a giant log-linear model will be built like this:

$$p(\underline{s} | \underline{x}; \underline{w}) = \frac{\exp(\underline{w} \cdot \Phi(\underline{x}, \underline{s}))}{\sum_{\underline{s}' \in S^m} \exp(\underline{w} \cdot \Phi(\underline{x}, \underline{s}'))}$$

Note: The set of all possible state sequence is  $S^m$ .  $\Phi$  is a feature vector that maps an entire input sequence  $\underline{x}$  paired with an entire state sequence  $\underline{s}$  to some d-dimensional feature vector. It can be regarded as a “global” feature because it takes the entire state sequence into account. That is the reason why I choose CRF rather than MEMM.

My goal is to use gradient-based optimization methods to find  $\underline{w}^*$  which maximizes the log-likelihood function:

$$\underline{w}^* = \arg \max_{\underline{w} \in \mathbb{R}^d} \sum_{i=1}^n \log p(s^i | \underline{x}^i; \underline{w}) - \frac{\lambda}{2} \|\underline{w}\|^2$$

If do the partial derivatives, you will find it involves a sum over  $V_l^m$ , a huge set.

$$\frac{\partial}{\partial w_k} L(\underline{w}) = \sum_i \Phi_k(\underline{x}^i, \underline{s}^i) - \sum_i \sum_{\underline{s} \in S^m} p(\underline{s} | \underline{x}^i; \underline{w}) \Phi_k(\underline{x}^i, \underline{s}) - \lambda w_k$$

Fortunately, it can be computed efficiently using forward-backward algorithm, a dynamic programming algorithm. In this manner, the whole formula will be concentrated on computing the transition parameters, a  $V_l \times V_l$  transition matrix. After estimating the parameters, the next step is to handle the decoding problem: for a given input sequence  $\underline{x} = x_1 \dots x_m$ , we should find the most likely underlying state sequence under the model, that is:

$$\arg \max_{\underline{s} \in \mathcal{S}^m} p(\underline{s} | \underline{x}; \underline{w})$$

This expression can be further simplified as follows:

$$\arg \max_{\underline{s} \in \mathcal{S}^m} p(\underline{s} | \underline{x}; \underline{w}) = \arg \max_{\underline{s} \in \mathcal{S}^m} \sum_{j=1}^m \underline{w} \cdot \underline{\phi}(\underline{x}, j, s_{j-1}, s_j)$$

Here is an intuition that  $\underline{w} \cdot \underline{\phi}(\underline{x}, j, s_{j-1}, s_j)$  represents an associated score for each transition from state  $s_{j-1}$  to state  $s_j$ . The score will be relatively high if the state transition is plausible, relatively low if this transition is implausible. Recall out previous step, we have already obtained the transition matrix. Thus, the decoding problem is using this transition matrix to find an entire sequence of states such that the sum of transition score is maximized. Again, this problem can be solved by using dynamic programming algorithm, the Viterbi algorithm.

Compare to previous beam search algorithm, the computational complexity of Viterbi algorithm is  $O(V^2N)$ , while spatial complexity is  $O(VN)$ .

Here is the pseudo-code of training procedure of the baseline model:

### Algorithm 1 Bidirectional-LSTM-CRF model training procedure without pos

```

corpus_input ← read_train_data_from_file
while batch has next :
     $x_i \leftarrow \text{embedding}(\text{corpus\_input}_i) \ i = 1 \dots t$ 
     $O_1 \leftarrow \text{bidirectional}(x_i)$ 
     $\text{logits} \leftarrow W_i O_i + b$ 
     $\text{likelihood, paramater} \leftarrow \text{CRF}(\text{logits})$ 
     $\text{loss} \leftarrow -\frac{1}{n} \sum_{i=0}^n \text{likelihood}_i, \ n = 0 \dots \text{batchsize}$ 
    use AdamOptimizer to update weight
    backward procedure
     $\text{viterbi\_sequence} \leftarrow \text{viterbi\_decode}(\text{logits, paramater})$ 

```

### Experiment 2: seq2seq-BiLSTM-CRF

Let's first back to the traditional sequence labeling. A bunch of carefully constructed features and language-specific knowledge resources could be widely used for solving this task. Figure 13 gives a list of standard features employed in state-of-art named entity recognition systems.

Feature	Explanation
Lexical items	The token to be labeled
Stemmed lexical items	Stemmed version of the target token
Shape	The orthographic pattern of the target word
Character affixes	Character level affixes of the target and surrounding words
Part of speech	Part of speech of the word
Syntactic chunk labels	Base phrase chunk label
Gazetteer or name list	Presence of the word in one or more named entity lists
Predictive token(s)	Presence of predictive words in surrounding text
Bag of words/Bag of N-grams	Words and/or N-grams occurring in the surrounding context

Figure13: Features commonly used in training named entity recognition systems

Recall that the given dataset not only include the raw data, but also have some preprocessed tagger, POS tagger and chunker. They both can be associated with each input as features (as shown in the figure13). But there are some

differences between these two taggers. If parsing a sentence, we would convert the sentence into a tree whose leaves will hold POS tags (which correspond to words in the sentence), but the rest of the tree would tell you how exactly these words are joining together to make the overall sentence. For example, an adjective and a noun might combine to be a 'Noun Phrase', which might combine with another adjective to form another Noun Phrase. Thus, a POS tagger can be thought of as a parser which only returns the bottom-most tier of the parse tree to you. A plain POS tagger is really fast but does not give you enough information. On the other hand, a chunker might be thought of as a parser that returns some other tier of the parse tree, which is a one of the approaches to shallow parsing, would divide a text into syntactically related group. It extracts only information about basic non-recursive phrases (e.g. verb phrases or noun phrases).

Meanwhile, NER usually consists of two steps: firstly, it finds chunks of named entities in the text (i.e. groups elements of the sequence); then it classifies each named entity into one of predefined type (Person, Location, etc). Apparently, chunking is closer to NER than POS tagging. Avoiding cheating, and making the neural network to its full potential, I choose the POS as the second input associated with raw input data to feed into the models. That inspired me to design a model included two sequences, one for raw input sequence, the other one for POS sequence.

The first comes to my mind is to use Sequence-to-Sequence (seq2seq) model, which have been successfully used for many sequential decision tasks such like machine translation, parsing, dialog generation and image captioning. (See figure 14)

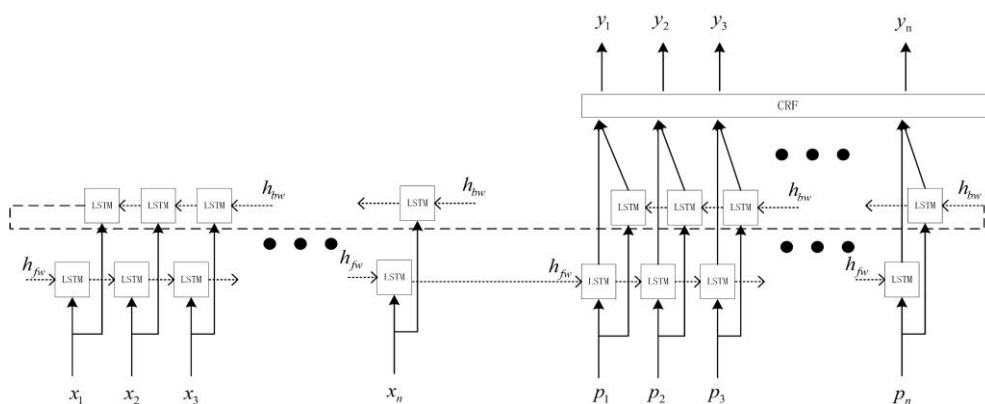


Figure 14: sequence-to-sequence BiLSTM-CRF

The idea is using one RNN to read the input sequence, one timestep at a time, to learn a dense black-box hidden representation of its sequential input. In my experiments, I also use RNN with BiLSTM activation unit for ameliorating the vanishing gradient problem when training long sequences. Typically, RNNs contain millions of parameters and utilize repeated transformations of large hidden representations under time-varying conditions. (See figure 15)

These hidden representations have proven to be very effective for classification. Note that LSTMs maintain both a cell state vector and a hidden state vector at each time step. It has been widely observed that the hidden states are able to capture important information about the structure of the input sentence, like discovering aspects of language such as phrases, grammar, or topics. After extract the features from the first input, I use second RNN connected to the last hidden state of the first RNN. In this way, the POS sequence is conditioned on the first input sequence. Then, the second RNN also have to learn hidden representations at each time-step like the first RNN, which are used for making decision of the distribution of the target labels. Finally, similar to previous model, by adding sentence level tag information via a CRF on top layer, the model can produce higher tagging accuracy. Here is the pseudocode to illustrate the training procedure:

Hyperparameters:

- Layer =n, hidden size =d
- Vocab for raw input data = $V_c$ , vocab for POS tagger = $V_p$ , vocab for target label set= $V_l$

First Sequence:

- Input: input embedding:  $|V_c| * d$
- Bi-LSTM:  $2n(8d^2 + 4d)$

Second Sequence:

- Input: input embedding:  $|V_p| * d$
- Bi-LSTM:  $2n(8d^2 + 4d)$

Output:

- output embedding:  $|V_l| * 2d$
- output bias:  $|V_l|$

Figure 15: illustrate the parameters

## Algorithm 2 Seq2Seq-BiLSTM-CRF model training procedure

---

```
corpus_input  $\leftarrow$  read_train_data_from_file
pos_input  $\leftarrow$  read_train_data_from_file
while batch has next :
     $x_i \leftarrow \text{embedding}(\text{corpus\_input}_i) \ i = 1 \cdots t$ 
     $p_i \leftarrow \text{embedding}(\text{pos\_input}_i) \ i = 1 \cdots t$ 
     $O_1, \text{state} \leftarrow \text{bidirectional}(x_i)$ 
     $O_2 \leftarrow \text{bidirectional}(p_i, \text{state})$ 
     $\text{logits} \leftarrow W_i T + b$ 
     $\text{likelihood}, \text{paramater} \leftarrow \text{CRF}(\text{logits})$ 
     $\text{loss} \leftarrow -\frac{1}{n} \sum_{i=0}^n \text{likelihood}_i, \ n = 0 \cdots \text{batchsize}$ 
    use AdamOptimizer to update weight
    backforward procedure
     $\text{viterbi\_sequence} \leftarrow \text{viterbi\_decode}(\text{logits}, \text{paramater})$ 
```

### Experiment 3: seq2seq-attention-BiLSTM-CRF

From previous model, we can see the second sequence is supposed to produce the final prediction at each time solely based on the last hidden state from the first sequence. Both cell state vector and hidden state vector of forward and backward direction respectively, must encode everything we need to know about the source sentence. It must fully capture its syntactic and semantic features. Thus, these vectors can be regarded as a sentence embedding. If you make an experiment by plotting the embeddings of different sentences in a low dimensional space using PCA or t-SNE, you can find that the semantically similar sentences end up cluster together.

But it seems somewhat unexplained to assume that we can extract all information about the first sequence into last hidden vector, especially for a potentially very long sentence. (In my project, the longest sentence has 113 words). Also, the two sequences are both well aligned, every word in the first sequence corresponds to a POS tagger sequentially in the second sequence. Let's say the first tagger in the POS sequence is highly correlated with the first word of the source sentence. That means each RNN unit in the second sequence has to consider information from 113 steps ago, and that information needs to be somehow encoded in the hidden vector. In order to deal with such long-range dependencies problems, we should find some approaches to dynamically select connection based on the raw input sequence and POS input sequence. Researchers have found that reversing the source sequence (feeding it backwards into the encoder) produces significantly better results because it shortens the path between these relevant parts. Similarly, feeding an input sequence twice also seems to help a network to better memorize things. However, these methods are not a principled solution. Here is an alternative method, attention mechanism, can handle this problem is a reasonable way in theory.

With an attention mechanism we no need to completely rely on the encoding the full source sentence into a fixed-length vector. Instead, we let the model learn what to attend to base on the input sentence and what the second sequence has produced so far. (See figure 16)

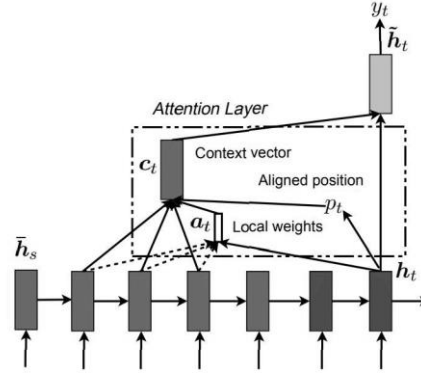


Figure 16: Attention Mechanism

Here, the  $h_t$  is the concatenated output of forward and backward results produced by the POS sequence bi-LSTM unit,  $\bar{h}_s$  are source sentence outputs also from the source sequence bi-LSTM units. The important part is that each output  $y_t$  now depends on a weighted combination of all the input states, not just the last state. The  $a_t$  is weight that define in how much of each input state should be considered for each output. Back to the previous example, the first tagger in the POS sequence will pay a lot of attention to the first state in the source sentence while producing the correspond label prediction, and so on. Noted that the  $a_t$ s are normalized to sum to 1, they are a distribution over the input states. Given the weights, the context vector  $c_t$  is computed as the weighted average over all the source hidden states. At last  $c_t$  will be associated with original output  $h_t$  to produce the final output  $\tilde{h}_t$ .

We can see attention comes at a cost. We need to calculate an attention value for each combination of source words and POS tagger. The previous example has a 113-word input sequence and a 113-word POS sequence that would be 113\*113 attention values which makes the computation more difficult and prohibitively expensive. Compare to previous model which has a lot of parameters (see figure 17), this model could be added 7 more parameters for the attention achievement. So, training network using backpropagation is relatively harder, though the mechanism here is soft.

$$a = \text{softmax}(a_v * \tanh(a_w_{\text{source}} * h_{\text{source}} + a_w_{\text{target}} * h_{\text{target}} + a_b))$$

- $a_w_{\text{source}}$ :  $2d * 2d$
- $a_w_{\text{target}}$ :  $2d * 2d$
- $a_b$ :  $1 * 2d$
- $a_v$ :  $1 * 2d$

$$\text{context} = a * h_{\text{source}}$$

$$h_{\text{target\_attend}} = \tanh(h_w_{\text{context}} * \text{context} + h_w_{\text{target}} * h_{\text{target}} + h_b)$$

- $h_w_{\text{context}}$ :  $2d * 2d$
- $h_w_{\text{target}}$ :  $2d * 2d$
- $h_b$ :  $1 * 2d$

Figure 17: seven more parameters for attention

After putting a CRF on the top layer, the whole model looks like this:

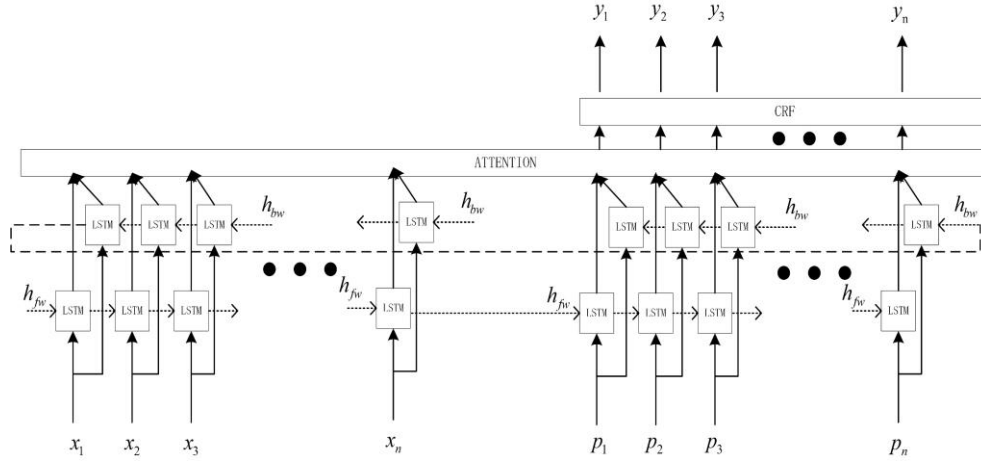


Figure 18: seq2seq+attention+lstm+crf

Here is the pseudocode:

### Algorithm 3 Seq2Seq-BiLSTM-Attention-CRF model training procedure

```

corpus_input ← read_train_data_from_file
pos_input ← read_train_data_from_file
while batch has next :
     $x_i \leftarrow \text{embedding}(\text{corpus\_input}_i)$   $i = 1 \dots t$ 
     $p_i \leftarrow \text{embedding}(\text{pos\_input}_i)$   $i = 1 \dots t$ 
     $O_1, \text{state} \leftarrow \text{bidirectional}(x_i)$ 
     $O_2 \leftarrow \text{bidirectional}(p_i, \text{state})$ 
     $T \leftarrow \text{attention\_layer}(O_1, O_2)$ 
     $\text{logits} \leftarrow W_i T + b$ 
     $\text{likelihood}, \text{paramater} \leftarrow \text{CRF}(\text{logits})$ 
     $\text{loss} \leftarrow -\frac{1}{n} \sum_{i=0}^n \text{likelihood}_i$ ,  $n = 0 \dots \text{batchsize}$ 
    use AdamOptimizer to update weight
    backforward procedure
     $\text{viterbi\_sequence} \leftarrow \text{viterbi\_decode}(\text{logits}, \text{paramater})$ 

```

### Experiment 4: seq+seq-BiLSTM-CRF

Actually, the attention mechanism is quite counterintuitive, not really like human attention. Our memory system is divided into working memory, short-term memory and long-term memory. The LSTM unit is designed to imitate the human short-term and long-term memory. On the other hand, working memory contains information about the focus of our attention. As the capacity of our working memory is rather small (research shows the capacity can range between 5–7 unrelated concepts), our attention is considerably selective. Our brain is simply not able to process all that is happening around us at once. It is instead narrowing down its focus to the most relevant pieces of information. Therefore, by focusing on one thing, we can neglect many other things. But that's not really what we're doing in the above model. We're essentially looking at everything in detail before deciding what to focus on. Intuitively that's equivalent generating an output, and then going back through all of internal memory of the text in order to make a



decision. That seems like a waste, and not at all what humans are doing. In fact, it's more akin to memory access, not attention. In human brain, the relevancy is determined by our own objectives which inspired me to design a new model, like this: (see figure 19)

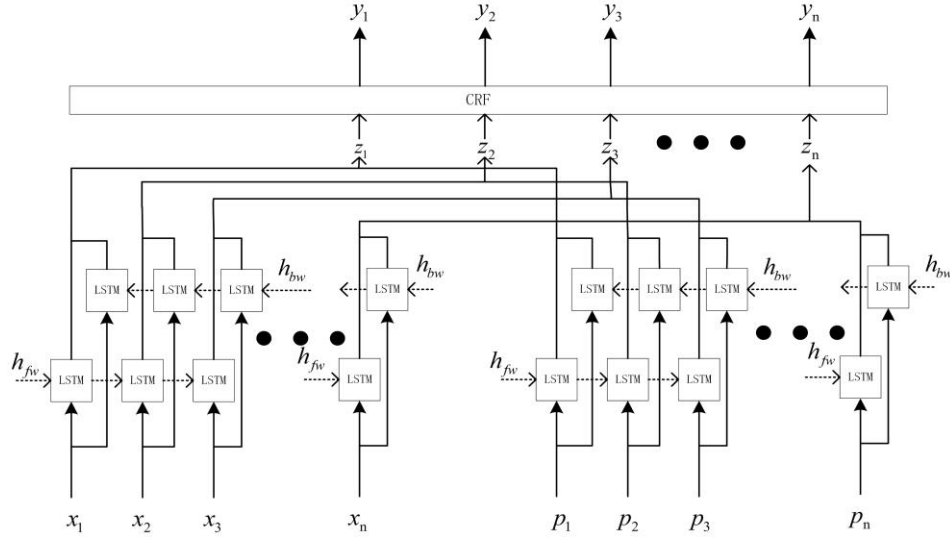


Figure 19: seq+seq-BiLSTM-CRF

The main idea is to obtain the output from source input and POS input independently, rather than using the last hidden state vector to transfer the features from first source sequence to the second POS sequence. Then I concatenated the source output with POS output explicitly, instead of only outputting the POS part. The rest of the architecture is the same as the previous models like using bi-directional LSTM and CRF, except the output embedding is  $|V_l| * 4d$ . Because the outputs are two 2d-dimension vectors concatenated with each other. The rationale of building the model like this is based on the observation that the source input sequence and POS tagger sequence are one-to-one correspondent. Intuitively, I can establish their dependency in advance, no need to use attention mechanism to compute the similarity. Seq2Seq is more like a linear workflow, while seq+seq is parallel operation which means it works more effective and without adding more parameters.

Here is the pseudocode:

#### Algorithm 4 Seq+Seq-BiLSTM-CRF model training procedure

```

corpus_input ← read_train_data_from_file
pos_input ← read_train_data_from_file
while batch has next :
     $x_i \leftarrow \text{embedding}(\text{corpus\_input}_i) \ i = 1 \dots t$ 
     $p_i \leftarrow \text{embedding}(\text{pos\_input}_i) \ i = 1 \dots t$ 
     $O_1 \leftarrow \text{bidirectional}(x_i)$ 
     $O_2 \leftarrow \text{bidirectional}(p_i)$ 
     $O \leftarrow \text{concat}(O_1, O_2)$ 
     $\text{logits} \leftarrow W_i O_i + b$ 
     $\text{likelihood, paramater} \leftarrow \text{CRF}(\text{logits})$ 
     $\text{loss} \leftarrow -\frac{1}{n} \sum_{i=0}^n \text{likelihood}_i, \ n = 0 \dots \text{batchsize}$ 
    use AdamOptimizer to update weight
    backforward procedure
     $\text{viterbi\_sequence} \leftarrow \text{viterbi\_decode}(\text{logits, paramater})$ 

```

## Evaluation

The familiar metrics of recall, precision and  $F_\beta$  measure are used to evaluate NER systems.

When  $\beta = 1$ , precision and recall are equally balanced; this is sometimes called  $F_1$ :

$$F_1 = \frac{2PR}{P+R}$$

We have three baselines:

One is the BI-LSTM-CRF model which was first applied by Zhiheng Huang team. This model can achieve the best F1 score of 90.10 with both Senna embedding and gazetteer features. Also, it is robust and has less dependence on word embeddings. It can produce accurate tagging performance without resorting to word embedding. Besides, the performance of other systems for NER is shown in figure 20. In the NER CoNLL 2003 challenge, the best performance has been obtained with 88.76% F1 score by a combined learning system that used Maximum Entropy Models, transformation-based learning, Hidden Markov Models as well as robust risk minimization (Florian et al., 2003). The best F1 score of 90.90% was reported in (Passos et al., 2014) which employed a new form of learning word embeddings that can leverage information from relevant lexicons to improve the representations.

System	accuracy
Combination of HMM, Maxent etc. (Florian et al., 2003)	88.76
MaxEnt classifier (Chieu., 2003)	88.31
Semi-supervised model combination (Ando and Zhang., 2005)	89.31
Conv-CRF (Collobert et al., 2011)	81.47
Conv-CRF (Senna + Gazetteer)	89.59
CRF with Lexicon Infused Embeddings (Passos et al., 2014)	90.90
BI-LSTM-CRF	84.26
BI-LSTM-CRF (Senna + Gazetteer)	90.10

Figure 20: Comparison of F1 scores of different models for NER

The second one is two neural architectures implemented by Lample, G. team -- one based on bidirectional LSTMs

Model	Variant	F <sub>1</sub>
LSTM	char + dropout + pretrain	89.15
LSTM-CRF	char + dropout	83.63
LSTM-CRF	pretrain	88.39
LSTM-CRF	pretrain + char	89.77
LSTM-CRF	pretrain + dropout	90.20
LSTM-CRF	pretrain + dropout + char	<b>90.94</b>
S-LSTM	char + dropout	80.88
S-LSTM	pretrain	86.67
S-LSTM	pretrain + char	89.32
S-LSTM	pretrain + dropout	87.96
S-LSTM	pretrain + dropout + char	90.33

Figure 21: English NER results, using different configurations. “pretrain” refers to models that include pretrained word embeddings, “char” refers to models that include character-based modeling of words, “dropout” refers to models that include dropout rate.

and CRFs, and the other that constructs and labels segments using a transition-based approach inspired by shift-reduce parsers. They use both pre-trained word representations and “character-based” representations that capture morphological and orthographic information. To prevent the learner from depending too heavily on one representation class, dropout is used. The highest  $F_1$  score is 90.94 for the LSTM-CRF model without any hand-engineered features or gazetteers. The transition-based algorithm performs less well than the LSTM-CRF model. (See figure 21)

The third baseline is the BiLSTM-CRF model implemented in my experiment. Unlike previous models, I neither use pre-trained word embeddings nor gazetteer features, and the  $F_1$  score is 91.02. Based on this weakest baseline, I experimented Seq2Seq-BiLSTM-CRF model by adding another input feature POS tagger, as the second sequence. This model shows the effectiveness by reaching a higher  $F_1$  score of 95.68. Then I used attention mechanism to allow the POS sequence attend sequentially to each input state, but the result is too bad. At some point, the attention is a misnomer. As I mentioned above, the mechanism is simply giving the network access to its internal memory and choosing what to retrieve from the memory rather than choosing what to attend to. That means the network retrieves a weighted combination of all memory locations, not a value from a single discrete location.

Model	cost	precision	recall	F1
BiLSTM-CRF (baseline)	2.85	95.29	87.11	91.02
Seq2seq-BiLSTM-CRF	1.43	96.79	94.59	95.68
Seq2seq-BiLSTM-Attention-CRF	7.14	54.13	43.26	48.09
Seq+seq-BiLSTM-CRF	1.38	97.61	94.58	96.07

Figure 22: results of four models in this project

Moreover, the relation between POS tagger and correspondent source word is deterministic. Interpreted another way, even though sometimes part-of-speech tagging and syntactic chunking can be cast as a multi-way classification task deployed as a sliding-window labeler. The token next to be labeled and its preceding and following words which included in those context window are aligned orderly. While attention could mess up the order, distract to other locations. For example, the POS tagger NN would distribute more weights on all the words which are nouns in the source input sentence, not according to its current location just give its window size source words more weights. Due to this disturbance, the training procedure became abnormally difficult. As a result, the  $F_1$  score is as low as 48.09. Finally, the new model Seq+Seq-BiLSTM-CRF, changing the previous structure to a large extent without sequence-to-sequence and attention mechanism, gave a more rational solution. Just as expected this final model outperformed other models with  $F_1$  score of 96.07, leading to the best tagging performance.

For the beginning example, I use my best implemented model to get the output as below:

*[LOC LONDON] 1996-12-06 [MISC Dutch] forward [PER Reggie Blinker] had his indefinite suspension lifted by [ORG FIFA] on Friday and was set to make his [ORG Sheffield Wednesday] comeback against [ORG Liverpool] on Saturday. [PER Blinker] missed his club's last two games after [ORG FIFA] slapped a worldwide ban on him for appearing to sign contracts for both Wednesday and [ORG Udinese] while he was playing for [ORG Feyenoord]*

Compare to the correct tagging, only the *Wednesday* in the last sentence failed to label, which is as same result as the Stanford NE recognizer.

Besides, here are some charts to show the test results for the average loss, precision, recall and  $F_1$  score:

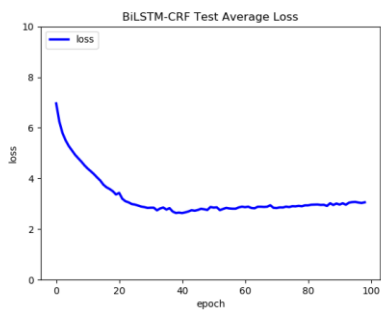


Figure 23.1 BiLSTM-CRF Test Avg Loss

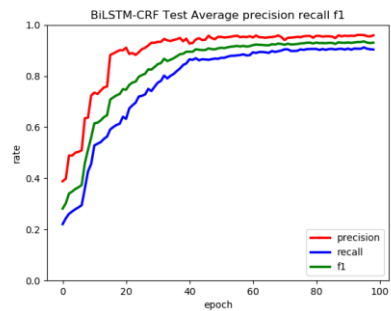


Figure 23.2 BiLSTM-CRF Test Avg Metrics

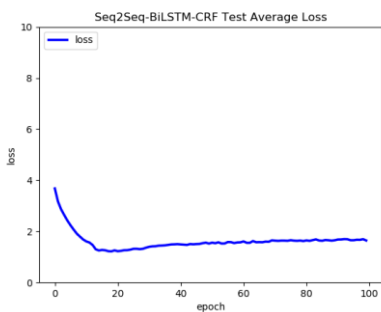


Figure 23.3 Seq2Seq-BiLSTM-CRF Test Avg Loss

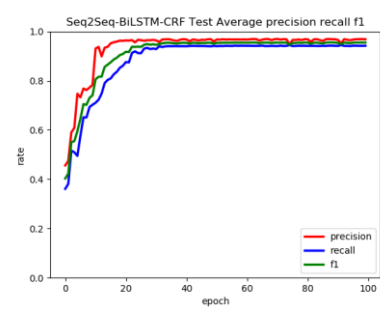


Figure 23.4 Seq2Seq-BiLSTM-CRF Test Avg Metrics

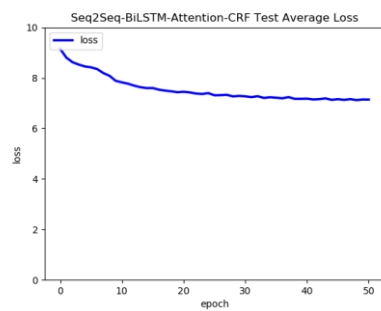


Figure 23.5 seq2seq-BiLSTM-Att-CRF Test Avg Loss

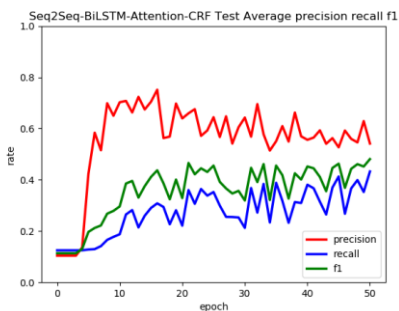


Figure 23.6 seq2seq-BiLSTM-Att-CRF Test Avg Metrics

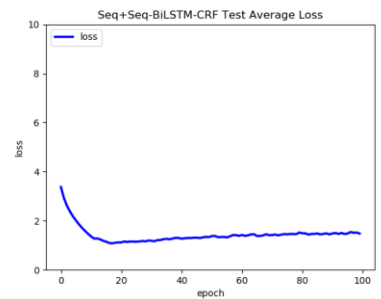


Figure 23.7 seq+seq-BiLSTM-CRF Test Avg Loss

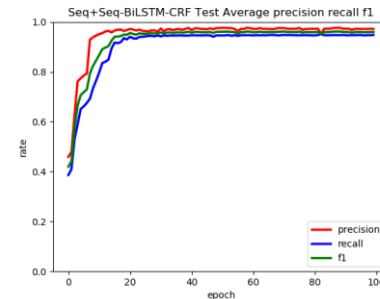


Figure 23.8 seq+seq-BiLSTM-CRF Test Avg Metrics

## Relevant background

All existing traditional sequence tagging models need the extraction of designed features which are then fed to a standard classification algorithm, for example, a Support Vector Machine (SVM), often with a linear kernel. In other words, only leveraging a large body of linguistic knowledge, researchers themselves can discover intermediate representations by engineering task-specific features. These features are often derived from the output of preexisting systems, leading to complex runtime dependencies. Therefore, the desire to avoid task-specific engineered features became the motivation of Collobert's team (Collobert et al., 2011) using a single learning system to discover adequate internal representations by transferring intermediate representations discovered on vast amounts of mostly unlabeled training data. Collobert named this approach as "almost from scratch" to emphasize the reduced reliance on a priori NLP knowledge. This unified neural network architecture and learning algorithm can be applied to various natural language processing tasks including part-of-speech tagging, chunking, named entity. Figure 24 gives the performance of state-of-the-art systems on four NLP tasks which is reported in accuracy for POS, and F1 score for CHUNK, NER and SRL. The last column displayed the performance of the engineered sweet spot (SENNA) on various tagging tasks.

Task	Benchmark	Data set	tags	Accuracy or F1	SENNA
POS	Toutanova et al. (2003)	WSJ	45	97.24%	97.29%
Chunking	Sha and Pereira (2003)	WSJ	42 (IOBES)	94.29%	94.32%
NER	Ando and Zhang (2005)	Reuters	17 (IOBES)	89.31%	89.59%
SRL	Koomen et al. (2005)	WSJ	186 (IOBES)	77.92%	75.49%

Figure 24: Performance of SENNA on four NLP tasks.

"SENNA" (Semantic/syntactic Extraction using a Neural Network Architecture) is a radically different approach compared to previous traditional methods. First, the system only uses rather simple input features during preprocessing and therefore avoids the nonnegligible computation time associated with complex handcrafted features. Second, it uses a multilayer neural network (NN) architecture, trained in an end-to-end fashion. The architecture takes the input sentence and learns several layers of feature extraction that process the inputs. The features computed by the deep layers of the network are automatically trained by backpropagation to be relevant to the task. All the networks were trained separately on each task using the sentence-level likelihood (the term SLL used in the original paper corresponds to training a CRF layer on the output). Third, most network computations are dense matrix-vector operations. In contrast, the system relies on a great number of sparse features experience memory latencies when traversing the sparse data structures. Finally, the compact implementation is self-contained. Since it does not rely on the outputs of disparate NLP system, it does not suffer from communication latency issues. The resulting system can be denoted as Conv-CRF model as it consists of a convolutional network and a CRF layer on the output.

## Reference

- Caputo, A., Basile, P., & Semeraro, G. (2009, September). Boosting a semantic search engine by named entities. In *International Symposium on Methodologies for Intelligent Systems* (pp. 241-250). Springer, Berlin, Heidelberg.
- Babych, B., & Hartley, A. (2003, April). Improving machine translation quality with automatic named entity recognition. In *Proceedings of the 7th International EAMT workshop on MT and other Language Technology Tools, Improving MT through other Language Technology Tools: Resources and Tools for Building MT* (pp. 1-8). Association for Computational Linguistics.
- Chinchor, N. A. (1998). Overview of muc-7/met-2. SCIENCE APPLICATIONS INTERNATIONAL CORP SAN DIEGO CA.
- Chinchor, N., Robinson, P., & Brown, E. (1998). Hub-4 Named Entity task definition version 4.8. Available by ftp from [www.nist.gov/speech/hub4\\_98](http://www.nist.gov/speech/hub4_98).
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug), 2493-2537.
- Doddington, G. R., Mitchell, A., Przybocki, M. A., Ramshaw, L. A., Strassel, S., & Weischedel, R. M. (2004, May). The Automatic Content Extraction (ACE) Program-Tasks, Data, and Evaluation. In *LREC* (Vol. 2, p. 1).
- Etzioni, O., Cafarella, M., Downey, D., Popescu, A. M., Shaked, T., Soderland, S., ... & Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1), 91-134.
- Evans, R., & Street, S. (2003). A framework for named entity recognition in the open domain. *Recent Advances in Natural Language Processing III: Selected Papers from RANLP*, 260(267-274), 110.
- Goyal, K., Neubig, G., Dyer, C., & Berg-Kirkpatrick, T. (2017). A Continuous Relaxation of Beam Search for End-to-end Training of Neural Sequence Models. *arXiv preprint arXiv:1708.00111*.
- Graves, A., Mohamed, A. R., & Hinton, G. (2013, May). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 IEEE international conference on* (pp. 6645-6649). IEEE.
- Grishman, R., & Sundheim, B. (1996). Message understanding conference-6: A brief history. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics* (Vol. 1).
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.
- Huang, Z., Xu, W., & Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*.
- Kawakami, K. (2008). Supervised Sequence Labelling with Recurrent Neural Networks (Doctoral dissertation, Ph. D. thesis, Technical University of Munich).
- Kazama, J. I., & Torisawa, K. (2008). Inducing gazetteers for named entity recognition by large-scale clustering of dependency relations. *Proceedings of ACL-08: HLT*, 407-415.
- Lafferty, J., McCallum, A., & Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025.

McCallum, A., Freitag, D., & Pereira, F. C. (2000, June). Maximum Entropy Markov Models for Information Extraction and Segmentation. In *Icml* (Vol. 17, No. 2000, pp. 591-598).

Mesnil, G., He, X., Deng, L., & Bengio, Y. (2013, August). Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech* (pp. 3771-3775).

Pascanu, R., Mikolov, T., & Bengio, Y. (2013, February). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning* (pp. 1310-1318).

Passos, A., Kumar, V., & McCallum, A. (2014). Lexicon infused phrase embeddings for named entity resolution. arXiv preprint arXiv:1404.5367.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

Rau, L. F. (1991, February). Extracting company names from text. In *Artificial Intelligence Applications, 1991. Proceedings., Seventh IEEE Conference on* (Vol. 1, pp. 29-32). IEEE.

Santos, D., Seco, N., Cardoso, N., & Vilela, R. (2006). Harem: An advanced ner evaluation contest for portuguese. In *quot; In Nicoletta Calzolari; Khalid Choukri; Aldo Gangemi; Bente Maegaard; Joseph Mariani; Jan Odjik; Daniel Tapias (ed) Proceedings of the 5 th International Conference on Language Resources and Evaluation (LREC'2006) (Genoa Italy 22-28 May 2006)*.

Sekine, S., & Isahara, H. (2000, May). IREX: IR & IE Evaluation Project in Japanese. In *LREC* (pp. 1977-1980).

Shen, H., & Sarkar, A. (2005, May). Voting between multiple data representations for text chunking. In *Conference of the Canadian Society for Computational Studies of Intelligence* (pp. 389-400). Springer, Berlin, Heidelberg.

Shinyama, Y., & Sekine, S. (2004, August). Named entity discovery using comparable news articles. In *Proceedings of the 20th international conference on Computational Linguistics*(p. 848). Association for Computational Linguistics.

Strobelt, H., Gehrmann, S., Pfister, H., & Rush, A. M. (2018). Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1), 667-676.

Tjong Kim Sang, E. F., & De Meulder, F. (2003, May). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4* (pp. 142-147). Association for Computational Linguistics.

Xu, P., & Sarikaya, R. (2013, December). Convolutional neural network based triangular crf for joint intent detection and slot filling. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on* (pp. 78-83). IEEE.

Yao, K., Peng, B., Zweig, G., Yu, D., Li, X., & Gao, F. (2014, May). Recurrent conditional random field for language understanding. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on* (pp. 4077-4081). IEEE.