Named Entity Recognition Homework 1

Francesco Starna
starna.1613660@studenti.uniroma1.it

Abstract

Named Entity Recognition (NER) is a sub-task of NLP which consists of recognizing named entities mentioned in unstructured text, such as persons, locations, organisations, quantities, etc. In this paper I will show how the task can be solved with a machine learning approach based on LSTM networks. I developed two models: one makes use of a large word-based representation while the other focuses on a character-based representation.

1 Introduction

NER is a challenging learning problem also in supervised approaches, because the training data are mostly unbalanced, making the learning process difficult and inefficient. The task to be solved is predicting four specific labels in a text corpora: organization (**ORG**), person (**PER**), location (**LOC**), and other (**O**). For example, given the following sentence

John went to California to visit Google.

the labels to be assigned are

PER O O LOC O O ORG

The two models have been designed to enhance different features: the LSTM-W model takes into account the complete representation of a word without any alteration (as is), while the LSTM-CW model takes the lower capitalization of the word, at the word level, and the complete representation of characters, at the character level. The reasoning behind these choices are intended to capturing the whole word meaning in each model, respectively.

2 Preprocessing

This phase is one of the most important, in fact, a lot of effort and different tries have been done in order to find the best approach. The aim of this step is to take the text corpora as input, build a vocabulary and give as output an iterable dataset, ready to fit the LSTM network.

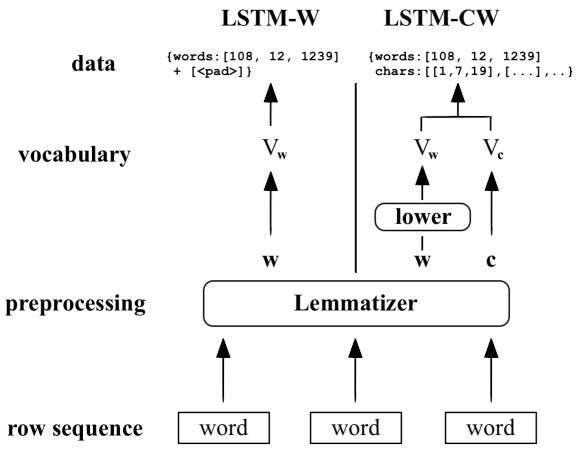


Figure 1: Preprocessing scheme. A distinction between LSTM-W model and LSTM-CW model can be clearly seen.

2.1 Filtering

The first thing I have worked on, is to apply a filter to the input sequences, in order to make the sequence more compact and build a smarted and richer (in a meaning sense) vocabulary. To do this, I used *lemmatization*. Let's first show an en explanatory example. If the input sequence is

Mario has bought new shoes for running.

then, the lemmatized input sequence becomes

Mario have buy new shoe for run.

We basically performed a text reduction; verbs are modified into their infinite form, and plural words into their singular form. This choice was quite important because it increased mostly the precision of PER, LOC, and ORG tags. This is due to

the fact that similar sentences (singular and plural words, many forms of the same verb, etc.) are, after lemmatization, balanced, and encourage the LSTM to learns better word belonging to the less frequent tags.

2.2 Vocabulary

Vocabularies are the object needed to encode each word to its number representation. Also here I make the decision to distinguish between the models.

The LSTM-W model builds a larger vocabulary, with each word taken as is from the training data, after have been lemmatized. The choice of having a larger set of word is motivated by the fact that there is no distinction between words with first capitalized letter and ones with not.

The LSTM-CW model builds a 2-dimension vocabulary divided in: a smaller word vocabulary, with each word taken without distinction of the first capitalized letter; character vocabulary of 50 characters with distinction between capitalized letters and lower letters. The choice of 50, comes from an approximated filtering process based on the frequency of the letters in the text, deleting all the ones that appear less than a certain threshold. Since the work of this model is to enhance the character level representation besides the word one, we use a lower dimension of embedding for the word. Larger or smaller sizes of vocabulary have been tried, with slightly different results.

2.3 Dataset

Once vocabularies are built, we need to process the data from text, and encode each sequence. The encoding process is different between the models.

The LSTM-W model needs only the encoding of words, and the training process will be performed with batches in parallel, therefore padding is added to the data and ignored during the training phase, in order to match the longest sequence in the text. Each sequence is organized as follows:

- list of encoded words
- list of encoded tags

The LSTM-CW model needs words and characters of each word to be processed together, since in order to build the character level embedding we have to use another bidirectional LSTM. In order to do that, the training process is done sequentially, sequence by sequence, without the need of using padding. Each sequence is organized as follows:

- list of encoded words
- list of lists of encoded characters
- list of encoded tags

The figure 1 shows the whole preprocessing phase.

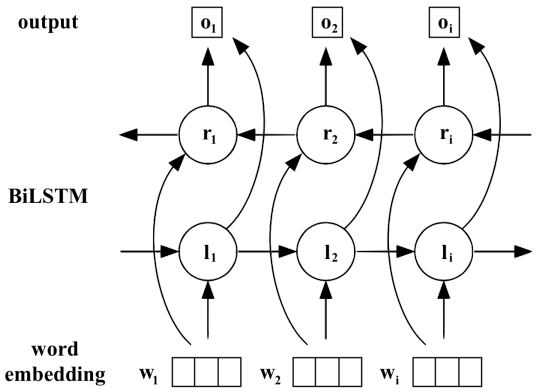


Figure 2: Model architecture. Each word is embedded in a vector representation and then passed to the bidirectional LSTM. $\mathbf{l_i}$ is the left context of the word, while $\mathbf{r_i}$ is the right context. The output $\mathbf{o_i}$ is a concatenation of the two layers.

3 Architecture

Recurrent neural network (RNNs) are the family of neural networks capable of work on sequential data, learning dependencies from sequences, and giving as output some information about them. The main problem is that they also suffer of bias from the most recent inputs. Long short term memory (LSTMs) have been introduced to solve this problem, in fact they effectively learn long inputs, controlling the proportion of data to process and the one to forget. The choice of LSTM was then obvious, since the dataset is quite large, divided into sequences, and we don't want to give bias to the model with respect to some text rather than other.

Both the models use bidirectional LSTMs, that process and combine the inputs in normal and reverse order, giving a more specific representation of a word (or character) within its context. The architecture is shown in figure 2.

3.1 Word Embedding

The embedding is the mapping between the vocabulary to each vector representation. In this paper, the

choice of the embeddings of each word is the core process, since, as we will see in the experiments results, there are important differences in the two models.

3.1.1 Pretrained Embedding

Both the models make use of pretrained word embeddings, specifically, global vectors for word representation (GloVe) (Pennington et al., 2014) of which reliability has been proved in many tasks. Global vectors are vector built according to word co-occurence probabilities on a text corpora. My choice was to use pretrained rather than self created embeddings, in order to avoid bias due to the smaller amount of training data available and to make the models competitive with different test data in several contexts.

3.1.2 LSTM-W

The word-based model takes 100-dimension GloVe vectors for embeddings, following the criteria of having a richer representation, with a larger vocabulary. Smaller dimension results in lower performance, while higher causes no important improvements.

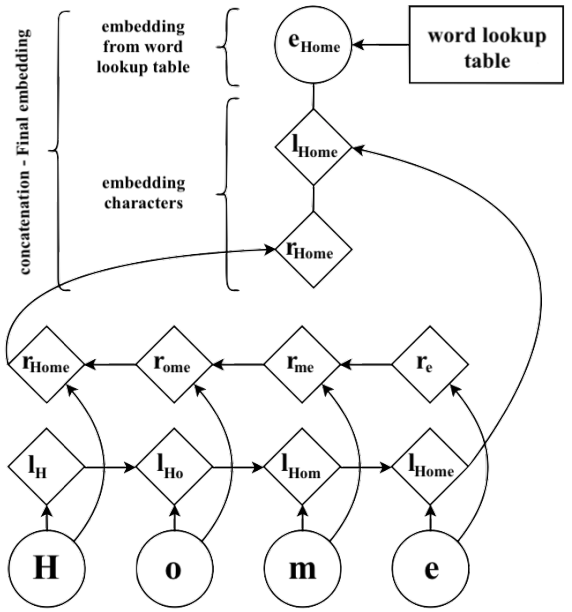


Figure 3: LSTM-CW Embedding. The character of the word 'home' are given to the bidirectional LSTM. Each character embedding is given by 1-hot encoding of the character vocabulary.

3.1.3 LSTM-CW

The character-based model is more complex. It builds the word embedding from two representations. The first comes from 50-dimension GloVe vectors. The second enhance the character representation of a single word, by running a bidirectional LSTM on the single characters. Every character of a word is given in normal and reverse order to a character Bi-LSTM, which returns a 50-dimension vector representation of the characters of a word. Now, we simply concatenate the two embeddings giving rise to a final 100-dimension vector word embedding. The reason behind this choice is the same of using LSTM networks; they learn sequences of data (no matter if are word or characters)

Another important detail in this implementation is the weight initialization of the character Bi-LSTM; since each word has its own character representation (capitalized letters, particular sequence of letters) it would have no sense capturing the meaning of all the words in a sequence. Therefore, the weights of the hidden state are initialized after each word, according to the He/Xavier normal distribution (Glorot and Bengio, 2010; He et al., 2015), which has demonstrated to be a good weight initialization function technique that tries to make the variance of the outputs of a layer to be equal to the variance of its inputs. Training the network with and without the weight initialization has showed significant differences in performance. Figure 3 shows the complete word embedding for LSTM-CW model.

3.2 Dropout

Initial experiments with both the model have been resulted in a poor performance. A dropout mask has improved performance; it is applied right after the word embedding of both the models; this has encouraged the LSTM-W model to preventing from overfitting and the LSTM-CW model to depend on both word and character representations.

4 Experiments

This section presents the methods used to train the models an the relative results obtained.

4.1 Training

Training is performed differently in the two models; while the LSTM-W model it has been processed in batches, the LSTM-CW one is processed sequentially. This obviously has a huge impact on the training time, because the second was ten times slower than the first one. The loss function is *Cross*

Entropy Loss, that combines log softmax and negative log likelihood. The optimization algorithm is also a crucial choice for the models, two of them, both stochastic, have been tried:

- Stochastic Gradient Descent (SGD) with gradient clipping 5 and learning rate of 0.1 (Lample et al., 2016).
- Adam (Kingma and Ba, 2017).

SGD is a standard optimizer, but it suffers from the exploding gradient problem; in order to overcome that, gradient clipping is proposed, which scale the gradient when it gets too large. Adam combines the benefits of RMSprop and SGD with momentum, resulting in a faster and ensured convergence.

4.2 Results

Table 1 presents the comparisons with the two models. The evaluation is performed through F1 score rather than accuracy, since the latter is not a good estimator because of the unbalanced data. Table 2, show the confusion matrices relative to highest score variants of the two models, respectively. It's clearly visible that the most frequent 'O' tag is of course the most right predicted. Regarding the other tags LSTM-W model predicts with a quite good precision (90%). That is not the case of the LSTM-CW model that performs good for 'O' and 'PER' tags, while 'ORG' and 'LOC' precision are quite lower (around 80%), mostly confused with the 'O' tag.

I was surpresed about the results, because I was expecting the LSTM-CW model to performs better in less frequent tags and worser in 'O' tag, with respect to the LSTM-W model, since most effort on enhancing the character representation of words was taken.

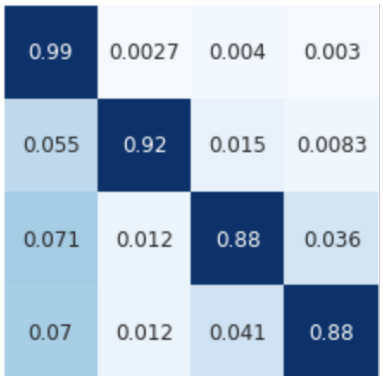
5 Conclusion

This paper presents the two models for NER, with their features and results, which are pretty good, considering the time given and that it was the really first attempt to build something real and complex. I am satisfied on the results all things considered, and I am sure that further improvements and implementation on the LSTM-CW model can results in a greater precision. One interesting approach might be CRF (conditional random field) which is suited in fields where information and representation of the neighbours affect the current prediction, like NER and more generally POS tagging.

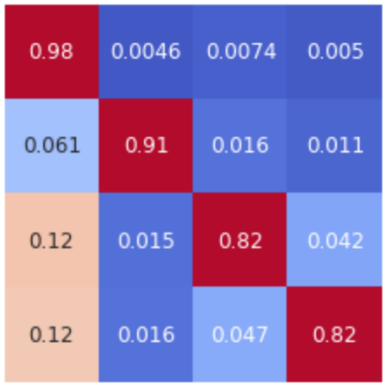
Model	Variant	F1
LSTM-W	pre50	0.84
	pre100	0.86
	pre100 + drop + sgd	0.89
	pre100 + drop + adam	**0.90**
LSTM-CW	pre50 + char	0.79
	pre50 + char + drop + sgd	0.81
	pre50 + char + drop + adam	**0.84**

Table 1: R1 score of both the models with different variants, where pre50/100 refer to the vector dimension of GloVe embeddings and drop to dropout mask.

Confusion Matrix



LSTM-W



LSTM-CW

Table 2: Confusion matrix is a performance measurement for machine learning classification problem where output can be two or more classes. Rows and columns refer to actual and predicted tags respectively. The tags O, PER, ORG, LOC (with respect to the given order), are the rows and column indices. The stronger the color, the more instances of class *y* (actual) are predicted as *x* (predicted).

References

Xavier Glorot and Yoshua Bengio. 2010. *Understanding the difficulty of training deep feedforward neural networks*. Universite de Montreal, Montreal, Canada.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. Microsoft Research.

Diederik P. Kingma and Jimmy Lei Ba. 2017. *Adam: a method for stochastic optimization*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. *Neural Architectures for Named Entity Recognition*. Carnegie Mellon University, Pittsburgh, Pennsylvania.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. *GloVe: Global Vectors for Word Representation*. Stanford University, Stanford, California.