Report homework NLP 2019/2020: Named Entity Recognition

Simone Antonelli
antonelli.1753685@studenti.uniroma1.it

Abstract

The problem to deal with for this homework is the Named Entity Recognition, an NLP task which aims to assign a named entity label to every word in a sentence. In particular, for the homework, there are four labels: ORG for Organization, LOC for Location, PER for Person and O for Other, respectively. To face this task I implemented a bidirectional LSTM with CRF layer, *i.e.* one of the model presented in the following paper (Lample et al., 2016).

1 Introduction

The task which consists to classify every word of a sentence in categories such as person, organization, location and others, is called Named Entity Recognition (NER). There are different ways to deal with this kind of task, like unsupervised learning and supervised learning approaches, and the one I chose is supervised learning. Using this approach, the NER task can be considered as a sequence labelling task. Initially, with this kind of approach were used hand-engineered features or specialized knowledge resources like gazetteers, but leveraging deep learning techniques to discover features automatically, improved the models until they reached the state-of-the-art.
Recurrent neural networks (RNNs) works very well in modelling sequential data, in particular, bidirectional RNNs can create a context using past information, retrieved from the forward states, and future information, retrieved from backward states, for every step in the input. Thus, each token of a sentence will have information about the whole sentence. But it has been proven that in practice RNNs tend to be biased from the most recent inputs. To avoid this behaviour can be used long short-term memory networks which can capture long-range dependencies. Moreover, since in NER task to predict the label of each word needs to make use of neighbour tag information, to decode it is used conditional random field.

2 Neural architecture

The chose neural architecture is composed of an embedding layer, followed by a bidirectional long short-term memory with a sequential conditional random layer.

2.1 Embedding layer

The input of the model is a vector which represents each word of the given sentence. The representation of every word is done by using an embedding layer which returns a fixed dimensionality real-vector that stores the semantic and syntactic information of words.
Starting with randomly initialized embedding I observed significant improvements with pre-trained ones. In particular, with pre-trained word embeddings, the model converges very quickly, allowing to get relevant results in fewer epochs as compared with the model which had the randomly initialized embeddings. I tried both Glove (Pennington et al., 2014), trained using Wikipedia 2014 and Gigaword 5, and fast text (Grave et al., 2018), trained on Common Crawl and Wikipedia, embeddings to observe that with the latter I get better performance.

2.2 Bidirectional LSTM

Long short-term memories (LSTM) are a variant of the RNNs that use memory cells to propagate information through the sequence without applying non-linearities (Eisenstein, 2019).
The reason for using the bidirectional LSTM is to create the context of each word in the sentence. To do this is used two different LSTM with different parameters. Indeed, the network takes as input a sentence represented as a sequence of n words, $(x_1, x_2, \ldots, x_n)$; each x_i of the sequence represents a word embedding of the i-th word of

the input sentence. This sequence is fed to a first LSTM which computes the left context of each token in the sentence, and then another LSTM reads the same sequence in a reverse way to create the right context of the tokens. These two representations are then concatenated to obtain the actual representation of the word in the context. This pair of LSTM is known as Bidirectional LSTM.

2.3 Conditional Random Field

Conditional random field (CRF) is a conditional probabilistic model for sequence labelling used when there are dependencies across output labels, like in NER task, since the grammar imposes constraints on the interpretable sequences of labels which are to be modelled jointly, and it is based on the score function:

$$\Psi\left(\vec{w}, \vec{y}\right) = \sum_{m=1}^{M+1} \psi\left(\vec{w}, y_m, y_{m-1}, m\right) \ . \qquad (1)$$

To estimate the parameters is needed to maximize the probability (equation 2) of a label given the word, which is based on the label of the previous word and the current word:

$$p(\vec{y} \mid \vec{w}) = \frac{e^{\Psi(\vec{w},\vec{y})}}{\sum_{\vec{y'} \in \mathcal{Y}(\vec{w})} e^{\Psi\left(\vec{w},\vec{y'}\right)}} \ . \qquad (2)$$

While to decode the best sequence of labels $\hat{y}$ which maximizes the previous probability

$$\hat{y} = \arg\max_{\vec{y}} \Psi\left(\vec{y}, \vec{w}\right) \qquad (3)$$

is computed efficiently using the Viterbi algorithm which leverages the dynamic programming technique for reusing work in recurrent computations. I implement the CRF layer taking inspiration by the following vectorized implementation.

3 Experiment

This section presents the model setup for the training phase and the outcomes and the workflow (illustrated in figure 1) of the latter.

3.1 Training process

To make the sentences readable by the neural network, each sentence is converted into a tensor of numbers each of which identifies a word in a vocabulary built from the training dataset. Each sentence is then padded to make their length homogeneous; this is useful to work with batches of sentences. These latter are input to the embedding layer which is initialized using fast text pre-trained embeddings which covers the 90% of the vocabulary, and the remaining embeddings are initialized randomly. The resulting of the embedding layer passed through a dropout layer, with $p = 0.5$, which drops out some unit to prevent overfitting and co-adaptations on training data. Then, tensors outputted by the previous layer are fed to the BiLSTM composed of one layer whose dimension is set to 128. Thereafter, the output of the LSTM is passed through another dropout layer and a dense layer which outputs the representation of the emission score to give in input to the CRF layer, together with the transition matrix obtained adding two new labels needed to represent the start and the end of the sentences, which returns the predictions.
Adam optimizer is used with a gradient clipping set to 2.0 (usually used to avoid exploding gradients in RNN). To prevent overfitting, besides the dropout layer, some hyperparameter tuning is done, in particular, the model works fine with learning rate set to $1\mathrm{e}{-2}$ and weight-decay set to $1\mathrm{e}{-7}$. The latter is also used to penalize big weights, and so penalize the complexity of the model.
To evaluate the model, the negative loglikelihood is used as loss function instead of likelihood as said in section 2.3 because compute the partial derivatives of the logarithm are easier than the partial derivatives of equation 2, and it must be minimized since minimizing the negative will yield the maximum.

3.2 Results

The metrics used to evaluate the model are the macro precision, macro f1 and macro recall that are computed independently for each class. The model is trained for 10 epochs where each epoch takes approximately 2 minutes, reaching about 88% of macro f1 on test dataset.
As can be seen in the confusion matrix in figure 2, most of the wrong predictions are for minority classes, *i.e.* the labels 'ORG' and 'LOC'; this is the reason of using macro metrics since they consider classes imbalance issue. The performance obtained on dev dataset are shown in table 1.

4 Conclusion

The resulting model is left very simple making it as generalised as possible so that it does not depend on the task, and still, it can achieve almost the performance of the state-of-the-art models.

Additional Resources

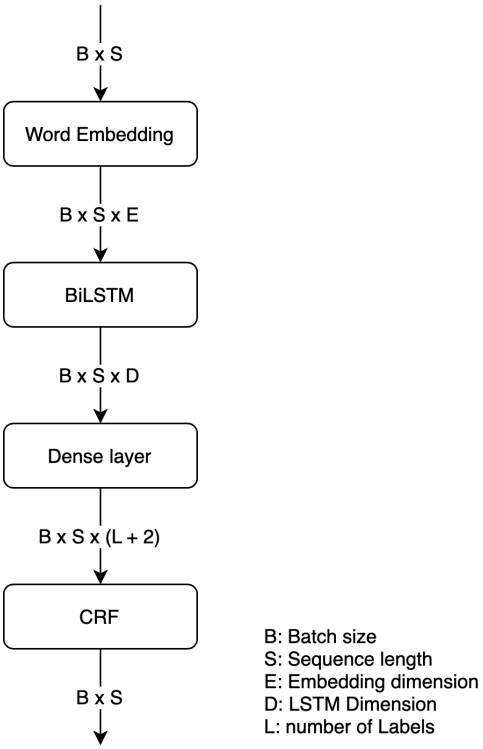

Figure 1: Training process workflow with tensors sizes.

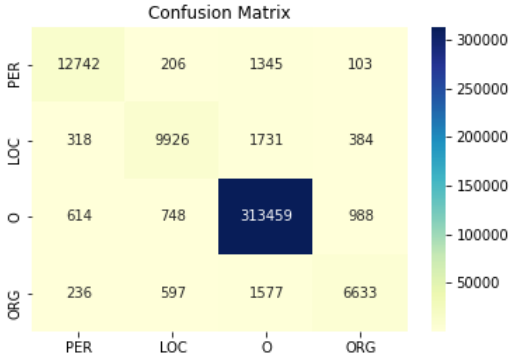

Figure 2: Confusion matrix represented as heatmap.

	precision	recall	f1-score	support
PER	0.92	0.89	0.90	14396
LOC	0.86	0.80	0.83	12359
O	0.99	0.99	0.99	315809
ORG	0.82	0.73	0.77	9043
accuracy			0.97	351607
macro avg	0.90	0.85	0.87	351607
weighted avg	0.97	0.97	0.97	351607

Table 1: Classification report on dev dataset.

References

J. Eisenstein. 2019. *Introduction to Natural Language Processing*. Adaptive Computation and Machine Learning series. MIT Press.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. Cite arxiv:1603.01360 Comment:Proceedings of NAACL 2016.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.