NLP Homework 1 - Report

Andrea Caciolai (1762906)

Abstract

In this document, I present the work I have done for the first homework of the Natural Language Processing course (A.Y 2019/2020), based on the Pytorch implementation of a BiLSTM model exploiting pre-trained word embeddings, character-level embeddings and POS tags to perform Named Entity Recognition.

1 Introduction

Named Entity Recognition (NER) is a classical problem in information extraction, with the core entity types of interest being people, locations and organizations (Eisenstein, chap. 8.3).

NER is generally formalized in the context of *sequence labeling* problems, those for which one cannot assign a tag to each token independently from the context of the token, but also on the context of the tag. So for a given sequence of tokens $\mathbf{w} = (w_1, w_2, \dots, w_M)$ there is a set of possible tags $\mathcal{Y}(\mathbf{w}) = \mathcal{Y}^M$, in which the labeling problem is a classification problem in the label space $\mathcal{Y}(\mathbf{w})$:

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\arg\max}\, \Psi(\mathbf{w}, \mathbf{y}), \tag{1}$$

where V is the vocabulary and for this homework $\mathcal{Y} = \{\text{'PER'}, \text{'LOC'}, \text{'ORG'}, \text{'O'}\}$ is the set of possible tags, while $\Psi(\mathbf{w}, \mathbf{y})$ is a *scoring function* from V^M to $\mathcal{Y}^M$ (Eisenstein, chap. 7).

For practical complexity reasons, this function must be decomposable into a sum of local parts:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \tag{2}$$

where each $\psi(\cdot)$ scores a local part of the tag sequence in reference to the sequence $\mathbf{w}$.

2 Model

With the framework introduced before, a classifier can be expressed as

$$\hat{\phi}(\mathbf{w}, y_m, y_{m-1}, m) = \mathbf{g}(\boldsymbol{\theta}; \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m)), \tag{3}$$

where $\mathbf{f}(\cdot)$ is the feature vector considering the entire input sequence $\mathbf{w}$, and also pairs of adjacent tags, thus modelling both token-tag features (emission features) and tag-tag features (transition features). An end-to-end model must learn both the weights for the classifier and an appropriate representation for the feature vector, able to capture the likelihood of a sequence of tokens being tagged with a sequence of labels.

2.1 Bidirectional LSTM

For this homework, I restricted myself to models able to capture only token-tag features, while higher expressivity can be achieved by considering both (Lample et al., 2016). In particular, I used a bidirectional LSTM model (BiLSTM), able to compute a representation $\mathbf{h}_t = [\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t]$ capturing both the left and right context of every token t in a given sequence. LSTMs are particularly good at retaining dependencies between tokens that are quite separated in the sentence, due to their gated architecture that addresses the problem of vanishing gradients, and for that, I decided to use this type of models instead of standard RNNs.

Since the BiLSTM model providing the feature vector $\mathbf{f}(\mathbf{w}, m)$ already contains several nonlinear activation functions in its memory cells, I chose a simple linear model

$$\hat{\phi}(\mathbf{w}, m) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, m) \tag{4}$$

for the classifier. In practice, this translates into a model having several embeddings layers (as explained in Section 2.2), followed by a deep BiLSTM layer, finally followed by a linear layer.

2.2 Embeddings

Since BiLSTM models are already provided within the Pytorch framework, my main contribution has been engineering good embeddings for the tokens in input. My work has been incremental, designing three different models of increased complexity of the embeddings they actuated on the incoming sequence of tokens.

I started with *pre-trained word embeddings*, which is the only embedding I used in the basic model. I could have obtained word embeddings by implementing a model like *word2vec* and train it on the given train corpus; however, both GloVe and FastText embeddings (the ones I experimented with) are obtained by training on corpuses much larger and richer and therefore I trusted their embeddings would be more expressive. So, the basic model embeds the tokens either with vectors of the pre-trained word embeddings, if present, or with random vectors otherwise.

However, this is a rather poor model for two reasons: the first and most evident is that a token not represented in the pre-trained word embeddings will be assigned to a random vector, carrying little to no relevant information able to help the model; the second is that as pointed out in (Lample et al., 2016) there is often a strong correlation between orthographic features of a token and its tag. Therefore I decided to capture these orthographic features in the form of *character embeddings*, that is the distinguishing feature of the second model. Since there is high variability in the length of the tokens, I chose an encoding scheme to represent tokens as a sequence of fixed-size characters, in particular consisting of $2k$ characters, with the first half being the first k characters and the last half being the last k characters of the token. My motivation is that to capture prefixes and suffixes, that often carry much of the orthographic information in language. Then, for each token, a second BiLSTM (trained along with the "main" one) extracts an embedding for a token starting from its character-level encoding sequence. This character-level embedding is then concatenated to the pre-trained word embedding, to have a final embedding

$$\mathbf{w}_t = [\mathbf{w}_t^{pre}; \overrightarrow{\mathbf{w}}_t^{char}; \overleftarrow{\mathbf{w}}_t^{char}] \tag{5}$$

for each token t in the sequence.

Finally, having captured token semantics with the word embeddings, orthographic features with the character embeddings, I tried to also capture syntactic behaviour by adding POS tags to the embedding of a token, obtained via a pre-trained POS tagger provided by NLTK.

2.3 Parametrization and training

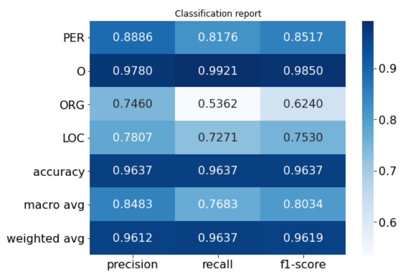
Both the training and evaluation (in Section 3) has been conducted on Google Colab, exploiting the GPU Tesla T4 with 16 GB of dedicated memory it provides. The parameters of the models outlined before, which are the BiLSTM and linear classifier ones, plus the ones for the non-pre-trained embeddings, are trained to minimize the *cross-entropy loss* between the predicted labels probability distribution and the ground truth ones, provided by the annotated training corpus. The models are all regularized via dropout to prevent co-adaptation, and the training is further regularized by means of early stopping based on the loss on a separate dataset, all to avoid overfitting, that in a very imbalanced dataset like the one of the homework would be severe. I have not employed other regularization techniques like *Lp-norm* weight decay, as several sources, such as (Pascanu et al., 2012), indicate their poor performance for training RNNs.

3 Results

In Fig. 1 and Fig. 2 we can see the classification reports and confusion matrices, respectively, computed on the provided test set for the three models.

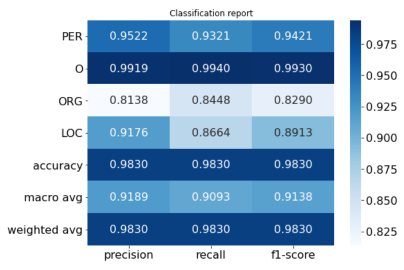
The character embeddings achieved a boost in performance of approximately 0.11, while the addition of also POS tags only brought about an increase of 0.060. It is interesting to note that the model was able to improve its precision on the most difficult tag ('ORG') at the cost of losing some of its recall on the same tag. An explanation for this behaviour might be that the information of the POS tag of a token brings more evidence to discriminate between tokens that are named entities and tokens that are not, while not offering any indication as to which type they are, exactly. This can be seen in the pie charts in Fig. 3 that show the strong correlation between all of the three named entities and the POS tag 'NNP'.

In any case, the final model achieves very good performances globally, with a macro f1-score of 0.9201. This performance is better than the one reported in (Lample et al., 2016) for the LSTM-CRF model they presented, even though that model, able to capture also transition features with its CRF layer, should be more powerful on paper.



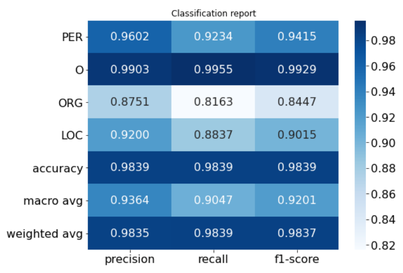
	precision	recall	f1-score
PER	0.8886	0.8176	0.8517
O	0.9780	0.9921	0.9850
ORG	0.7460	0.5362	0.6240
LOC	0.7807	0.7271	0.7530
accuracy	0.9637	0.9637	0.9637
macro avg	0.8483	0.7683	0.8034
weighted avg	0.9612	0.9637	0.9619

(a) Basic model



	precision	recall	f1-score
PER	0.9522	0.9321	0.9421
O	0.9919	0.9940	0.9930
ORG	0.8138	0.8448	0.8290
LOC	0.9176	0.8664	0.8913
accuracy	0.9830	0.9830	0.9830
macro avg	0.9189	0.9093	0.9138
weighted avg	0.9830	0.9830	0.9830

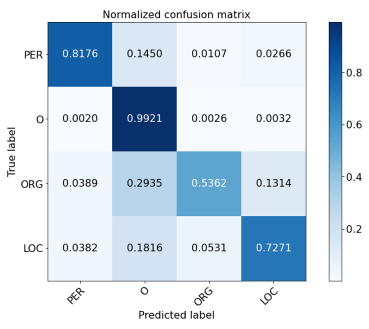
(b) Model with character embeddings



	precision	recall	f1-score
PER	0.9602	0.9234	0.9415
O	0.9903	0.9955	0.9929
ORG	0.8751	0.8163	0.8447
LOC	0.9200	0.8837	0.9015
accuracy	0.9839	0.9839	0.9839
macro avg	0.9364	0.9047	0.9201
weighted avg	0.9835	0.9839	0.9837

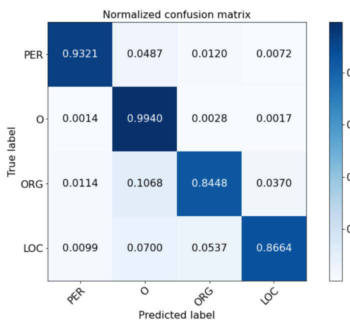
(c) Model with character and POS embeddings

Figure 1: Visualization of the classification reports for the three models.



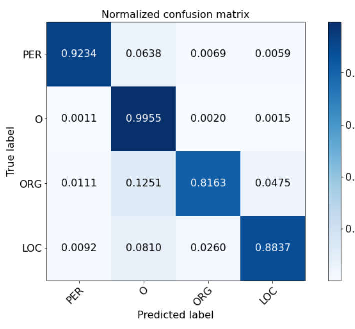
	PER	O	ORG	LOC
PER	0.8176	0.1450	0.0107	0.0266
O	0.0020	0.9921	0.0026	0.0032
ORG	0.0389	0.2935	0.5362	0.1314
LOC	0.0382	0.1816	0.0531	0.7271

(a) Basic model



	PER	O	ORG	LOC
PER	0.9321	0.0487	0.0120	0.0072
O	0.0014	0.9940	0.0028	0.0017
ORG	0.0114	0.1068	0.8448	0.0370
LOC	0.0099	0.0700	0.0537	0.8664

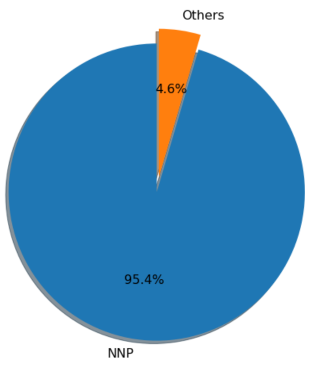
(b) Model with character embeddings



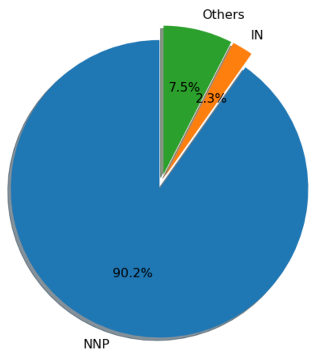
	PER	O	ORG	LOC
PER	0.9234	0.0638	0.0069	0.0059
O	0.0011	0.9955	0.0020	0.0015
ORG	0.0111	0.1251	0.8163	0.0475
LOC	0.0092	0.0810	0.0260	0.8837

(c) Model with character and POS embeddings

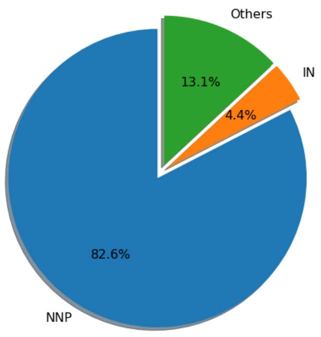
Figure 2: Visualization of the confusion matrix for the three models.



(a) Person



(b) Location



(c) Organization

Figure 3: Visualization of the distribution of POS tags for the three named entities.

A Additional Material

In Fig. 4 we can see a visualization of the embedding process, which makes data ready to be fed to the model by means of several embedding layers acting on the tensorized input.
The model (as a whole) receives three different tensors:

1. a tensor of shape $(B \times W)$, with B the batch size and W the window size, where the element i, j represents the index of the j-th token of the i-th sample in the batch, taken from the token vocabulary;

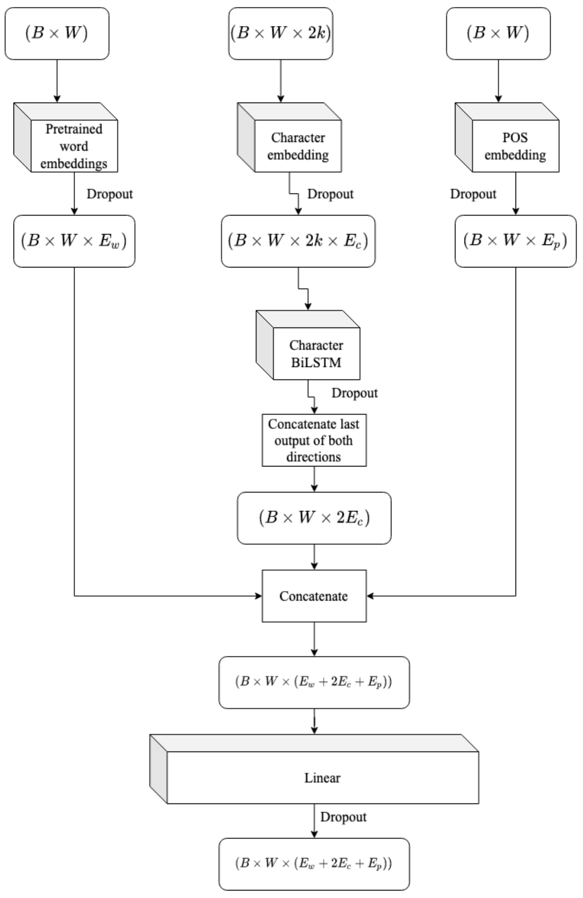
2. a tensor of shape $(B \times W \times 2k)$, with $2k$ the length of the character encoding sequence (as indices of characters) chosen to represent the j-th token of the i-th sample in the batch (k for the prefix, k for the suffix);

3. a tensor of shape $(B \times W)$, with the element i, j representing the POS tag (again represented as an index from a vocabulary) of the j-th tokens of the i-th sample in the batch.

These tensor containing indices get embedded, the first and last directly by embedding layers, while the second by the character level BiLSTM. Then the three embeddings get concatenated and are passed to a linear layer to obtain the best (in terms of performance of the subsequent BiLSTM model) linear combination of the information of the three embeddings combined.

In Table 1 I have reported the values for the hyperparameters I used for training the best version of the model described in Section 2.

I have put particular care into choosing the optimal values for the window size W and the character encoding length k. The data for this homework comes in the form of already tokenized sentences, so a list of list of tokens. To proceed with batched learning, the tokens in each sentence cannot be embedded directly, since then the length of the sequences sent to the model would vary across the samples of the batch. Therefore I introduced *windows*, splitting sentences longer than the window size into more windows, and padding the sentences (or split sentences) shorter than the window size with the special token '<pad>'. To choose the value for the window size I studied the distribution of the sentence lengths across the training set, as we can see in Fig. 5, taking then the mean of this distribution.



$(B \times W)$
$(B \times W \times 2k)$
$(B \times W)$
Pretrained word embeddings
Character embedding
POS embedding
Dropout
Dropout
Dropout
$(B \times W \times E_w)$
$(B \times W \times 2k \times E_c)$
$(B \times W \times E_p)$
Character BiLSTM
Dropout
Concatenate last output of both directions
$(B \times W \times 2E_c)$
Concatenate
$(B \times W \times (E_w + 2E_c + E_p))$
Linear
Dropout
$(B \times W \times (E_w + 2E_c + E_p))$

Figure 4: Embedding process scheme

Hyperparameter	**Value**
Window size (W)	25
Window shift	25
Word embedding dimension (E_w), pre-trained from FastText	300
Character encoding length ($2k$)	10
Character embedding dimension (E_c)	50
POS tag embedding dimension (E_p)	10
BiLSTM layers	2
Layers of BiLSTM for char embeddings	1
Dropout	0.5
Hidden dimension for the classifier (H)	128
Optimizer	*Adam*
Learning Rate	10^{-3}
Batch size	128

Table 1: Hyperparameter values used in training

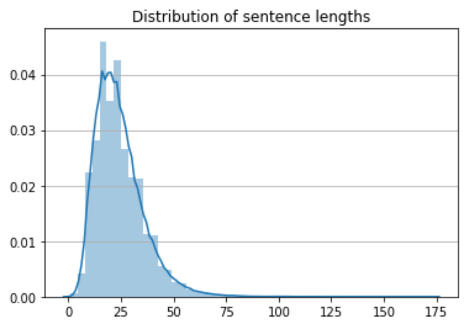


Figure 5: Distribution of the lengths of the sentences

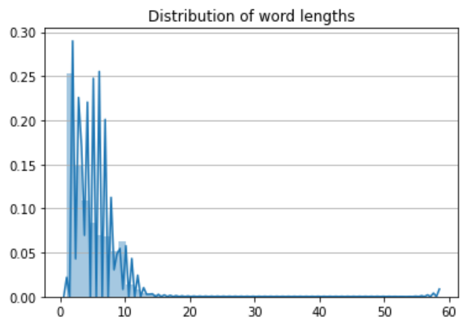


Figure 6: Distribution of the lengths of the words

As mentioned in Section 2.2, the same problem of homogeneity of sequence length came up also for deriving character level embeddings. To choose the number of characters to take to represent a token (to be precise, the value k referenced before), in a similar way I studied the distribution of the word lengths across the sentences in the training set, as we can see in Fig. 6, again taking the mean of the distribution.

The other hyperparameters have been set after performing some fine-tuning, starting from the "nominal" values set in the "Notebook #3 - POS Tagging" notebook Prof. Navigli presented in class and then tweaking them and observing the changes in performance, if any, retaining the ones yielding the best results.

References

Jacob Eisenstein. *Introduction to natural language processing*. The MIT Press.

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

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. On the difficulty of training recurrent neural networks.