NLP-2022 Homework 1: Named Entity Recognition

Dennis Rotondi
1834864
Sapienza, University of Rome
rotondi.1834864@studenti.uniroma1.it

1 Introduction

The first homework for the course NLP-2022 is about BIO-Named Entity Recognition, the task of tagging entities mentioned in different segments of text with their corresponding type. By nature the problem requires the analysis of highly unbalanced dataset Figure 1 and for this reason simple approaches like BoW miserably fail because do not take into account the context in which entities appear. I decided to start small and incrementally change the model according to the results. To embrace the use of NLP best practices I've based my work on what we've seen in class.

2 Method

The general idea for all the models I've developed follow the TAs' advised structure Figure 3 and the Pos Tagging's case study [2].

2.1 Vocabulary

There is no good model with a bad training set, and what establish its goodness here, more than in other NLP tasks due to the few entities in each sentence, is the adopted vocabulary. For this reason I commit myself to build the best one in terms of completeness/size ratio. I ended up downloading all the different pre-trained embeddings available on the Stanford site for Glove Vectors [3]. The training set is relatively small so (almost) every word is precious: I found out that combining Glove with Gensim [5] it's possible to cover many of the most common OOV entities for only the former. I've handled the ones still missing noticing that there was a common denominator: all of them appears with a punctuation eg "disney+","wars:","california," so I defined multiple special tokens associated to the most frequent punctuation to encode words where they appear instead of the simple <UNK>.

2.2 Preprocessing

Once I prepared the vocabulary, to increase consistency for the training, I've filtered out the "too dirty sentences", namely the ones that have all "O" labels or the "non-O" labels are for OOV words. Finally, using the same sliding window approach seen in [2] I've packed the sentences in a blocks of size 50 chosen according to [1] and the reduced scale of the task Figure 4; at the same time I encoded each syntagma in an index, associated to it when the vocab has been built, ready to be retrieved by an embedding layer (made using the pre-trained one) to output the correct representative vector.

2.3 LSTM

As stated the context cannot be neglected for our objective, before classification, starting from the representative vector, it's mandatory to encode the series of words (under the form of vectors) to have as many information as possible. It's here where come to the aid RNNs, the general idea is that it takes as input a sequence of vectors $x_1, ..., x_n$ and an initial state vector h_0 and it returns a list of hidden/state vectors $h_1, ..., h_n$ and a list of output vectors $\hat{y}_1, ..., \hat{y}_n$ where each $\hat{y}_i$ is a function of the corresponding state vector h_i: a kind of memory of the previous i inputs. To overcome the vanishing gradient problem of the basic architecture that would result in saturation of the state vectors during the backpropagation step, I decided to use as core of all my experiments the Long-Short Term Memory Networks (LSTM) [6]. Briefly this solution uses constant error carousels, which enforce a constant error flow within special cells; access to the cells is handled by multiplicative gate units, which learn when to grant access. Moreover it's possible to let information flow from the first word to the last and vice-versa, enabling this bidirectionality we have a Bi-LSTM, a key structure for my

method since increase contextualization.

2.4 Classifier

The last part of the model is a subnetwork that combines the priors from LSTM to assign the final labels. During the training processes the optimizer will try to minimize the classification error that consist in a cross-entropy probabilistic loss.

3 Experiments

What I've done after that all the pieces have been defined is to change every hyperparameters: starting from the embedding, passing through the optimizer and eventually for the model structure, extrapolating the context using one or more RNNs and classifying using different strategies, always with the aim of maximizing the scores on the dev-set and minimizing the overfitting. What follow next are some of the most interesting trials.

3.1 Baseline

As baseline I decided to use the solution of [2] with only the glove 100d embedding freezed, no preprocessing, a simple unidirectional LSTM with hidden size of 128 and a linear classifier on top, trained for 100 epochs using the default Adam optimizer. I've repeated the training for the same amount of epochs changing only the fact that this time embedding could be fine tuned. As Figure 2 shows these results just allow me to non fail the hw (0.45 F1), but there still is a huge margin of improvement.

3.2 A cool small model

Ascertain that the baseline scores poorly, before to add complexity to the model I decided to apply what I've described above in "Method" trying different configuration where I still had uncertainty like for the Glove dimension that, as expected, the greater the better. I also went for Bi-LSTM and used different hidden dimensions and numbers of recurrent layer. My significant experiments from the baseline are summarized in Table 1, only changing the hyperparameters and do not touching the baseline code I reach an impressive 0.73 F1 Score, giving me the hope that I still have opportunity for enhancement. Note that in these trials I've always used Adam interchanging lr={1e-3, 1e-4} with weight_decay=1e-6, while in general it gave me a better training than SGD I do not notice huge differences changing other parameters. In Figure 5 a comparison with the baseline: after all the adjustments, we have much better plots than at start.

3.3 Is complexity worth?

Even if the small model have a respectable F1 metric, I thought that the overall structure could be slightly changed to attain better performances. For this reason I first tried to extend the classifier using a MLP with the ReLU/tanh function for the non linearity part and then I attempt to enlarge the embedded sequence parallelly using a LSTM and a GRU: the concatenation of their outputs will be used as input for the final classifier. Unluckily both these solutions results in a failure <0.70.

3.4 Conditional Random Fields Layer

Last thing I wanted to try is a CRF layer [7] on top of the linear classifier that let to work with the conditional log likelihood of a sequence of tags which takes into account the label context, ideal to learn the "beginning" and "inside" displacement. On paper a really promising introduction nevertheless it just improve my average F1 of 0.01.

3.5 Torch.manual_seed(3407) is all you need

The positive side of using a simple model is that it's really fast to train and to exploit this I applied an empirical procedure described in [4], an article that highlight how the random seed can make the difference . I'm working with a 3080 RTX so leaving the computer on at night allows me to complete a hundred of training while automatically saving the best models. I did this for four days and what I observed have been many results >0.75 with a best of 0.7566. Another very crucial point is that from this process I got tons of weights coming from completely different initialisations and perfectly suit for an ensemble method.

4 Final Results

It's indeed with an ensemble of networks that I got my highest F1 of 0.7763. I want to conclude this report commenting just its confusion matrix Figure 6: here we have how many times an instances of $class_i$ (on row) is predicted as $class_j$ (on columns). It's clear that this solution predicts almost perfectly "O","B-per","I-per" labels respectively in 99%,93% and 94% of the cases, while struggle at most for "B-Prod" and "I-Prod" classes picking the correct one only 63 and 70 times out of 100. The main reason of this disparity is found in the training set, as seen in Figure 1 we have exactly less samples for the worst predicted classes.

Figures and Tables

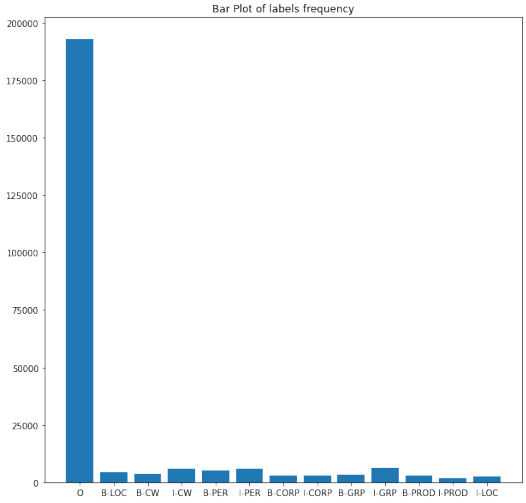

Figure 1: A bar plot built on the training set shows that, as expected, most of the words aren't named entities.

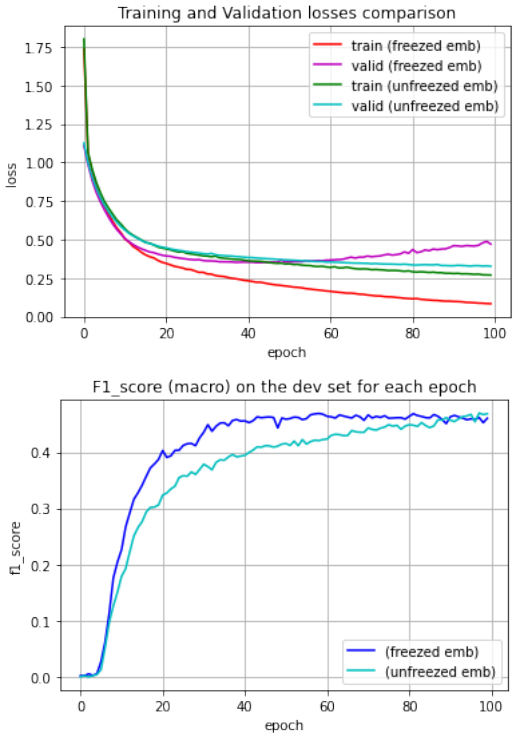

Figure 2: Plot of the loss and F1 score for the baseline model freezing and not freezing the embedding.

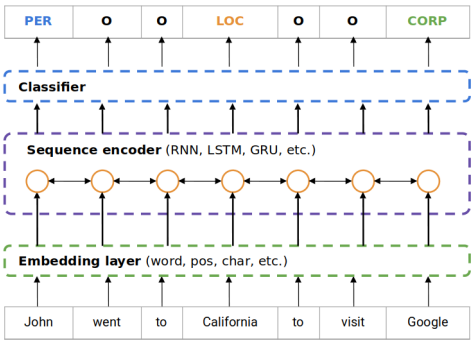

Figure 3: An advised approach: it's a model constitute of an embedding layer, sequence encoder and classifier.

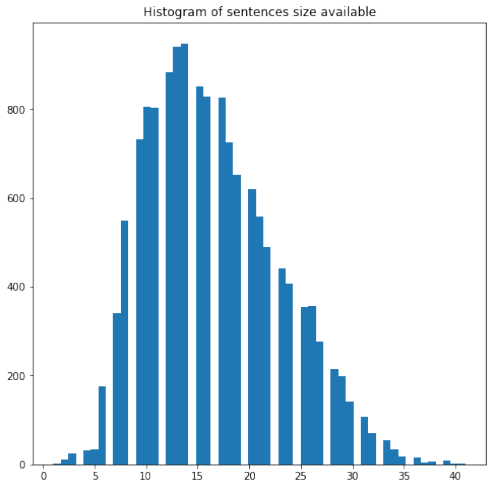

Figure 4: An histogram built on the length of each sentence, basically this distribution it's a Gaussian with mean $\sim$16 and std $\sim$6, confirming the small task nature.

Baseline	0.45
Change	**+F1**
glove.6B.200d	+0.02
preprocessing	+0.03
Bi-LSTM+dropout	+0.06
glove.6B.300d&gen	+0.05
270-HiddenSize	+0.07
5-RecurrentLayers	+0.05
Final Model	**0.73**

Table 1: Improvement of F1 after each change starting from baseline and training for $\sim$100 epochs.

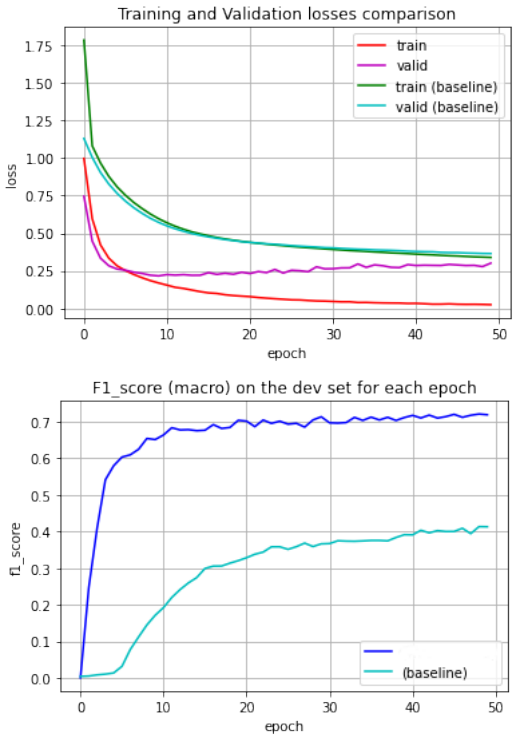

Figure 5: Loss and F1 of Baseline vs Fine-tuned model in a 50 epoch training.

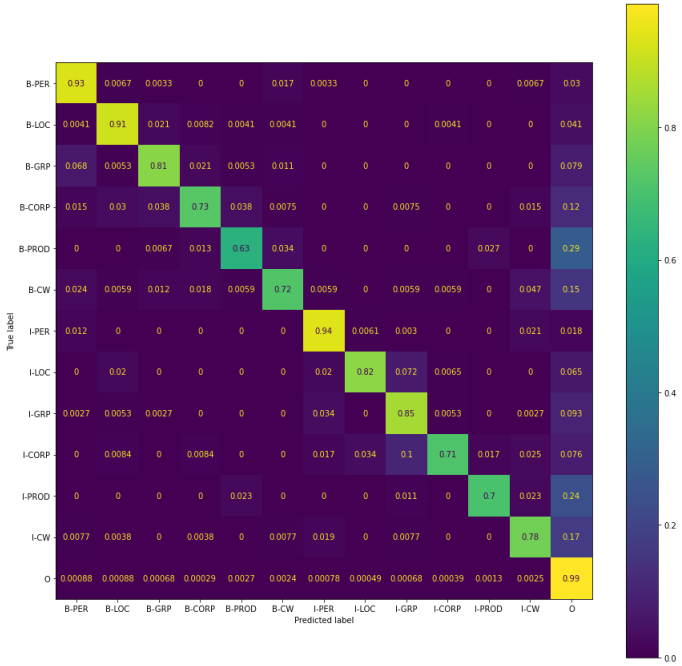

Figure 6: A confusion matrix of the ensemble method, the best I attain during my experiments. The more yellow a cell, the higher the chances to predict the row class as the column class.

References

[1] Gábor Borbély and András Kornai. 2019. Sentence length.

[2] Sapienza NLP research group. 2022. Notebook 6: Pos tagging.

[3] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[4] David Picard. 2021. Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision.

[5] Radim Rehurek and Petr Sojka. 2011. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2).

[6] Ralf C. Staudemeyer and Eric Rothstein Morris. 2019. Understanding lstm – a tutorial into long short-term memory recurrent neural networks.

[7] Charles Sutton and Andrew McCallum. 2010. An introduction to conditional random fields.