NLP homework 1: Named Entity Recognition

Giorgio Strano
strano.1809528@studenti.uniroma1.it

1 Introduction to the task

The objective was to perform automatic *Named Entity Recognition (NER)* on English text, classifying each word representing a named entity as one of the following categories: *Person, Location, Group, Product, Corporation* or *Creative Work*; using the *BIO* tag scheme for a total of 13 classes.

2 Dataset and metrics

The given dataset contained a few thousands English sentences, all in lowercase. Because of the very skewed distribution of labels, the plain accuracy metric was not very relevant, as even a model that always outputs *"O"* would achieve around 80%. Therefore, the main metric used for evaluation was the F1-score, as computed from the *seqeval* package (Nakayama, 2018). Almost no data preprocessing was needed, except to build a vocabulary to convert words into indices (or *one-hot vectors*), and to organize the words into batches of a fixed length (I used mostly 50 and 100-sized windows) with proper padding, to allow parallel data loading into fixed-size tensors.

3 The main model

Because of the sequential nature of the task, an appropriate neural model to solve it can be a Recurrent Neural Network. As my base model I used the combination of an embedding layer, a BiLSTM module, and a fully connected layer to classify. The embedding layer holds a trainable matrix containing a dense 100-dimensional vector for each word in the vocabulary. After a word has been embedded, it is fed to the Bidirectional Long-Short Term Memory module. This has a hidden size of 100, meaning that it outputs a 100-dimensional vector for each direction. Its output is then classified by the fully-connected layer. An important aspect I noticed is that the BiLSTM has to be tested applying padding to the sentences to make them the same size as those used during training: in fact, the right-to-left LSTM performs worse when it receives non-pad tokens in the rightmost end of a sentence, which never happens during training.

3.1 Word embeddings and vocabulary

A crucial choice affecting the performance of the model is how to embed the words as dense vectors. At first, I tried a simple embedding layer trained alongside the network: this used a vocabulary obtained from the training set by filtering out all the words that occurred less than 2 times, for a total of 11727 entries. It led the previously described model to a maximum F1-score of 60%. Then, I experimented with pretrained GloVe embeddings (Pennington et al., 2014), stored as a 400000×100 matrix. Here, the vocabulary was much bigger, and only 188 out of the 11727 words in the training set were missing. Having these embeddings preloaded and then fine-tuned during training caused a massive improvement, bringing the F1-score on the validation dataset to 69.65%.

4 Improvements to the baseline model

4.1 Part of speech tagging

After establishing a baseline model, I tried out different strategies to improve it. The first was to use a pretrained Part-Of-Speech tagger to add the POS of each token as a feature. To do so, I loaded a pipeline from STANZA (Qi et al., 2020) into a new dataset, which returned a list of POS-tags alongside the lists of words and labels. These tags were converted to 17-dimensional one-hot vectors, and concatenated to the word embedding before feeding them to the LSTM. This improved both models (the one with my own embeddings, and the one using GloVe vectors) by a negligible amount (to,

respectively, 61.37% and 69.79%).

4.2 More complex architecture

Since the model was not severely overfitting yet, I thought a more complex architecture might have been appropriate: in fact, I managed to improve to 70.71% F1-score by adding a second fully connected layer to the classifier with a 50% dropout in between, and raising the hidden dimension of the LSTM to 200. Later, I figured that a better way to regularize was to go back to my first architecture, adding instead a second stacked layer to the LSTM module, again with a 50% dropout, which improved the F1-score to 73.85%.

4.3 More feature extraction: Character embedding

As a last effort to improve the feature extraction aspect of the model and its accuracy on unknown tokens, I extended it with a character embedding mechanism. The basic idea is to set up an embedding layer to obtain a dense vector representation of each character, and then aggregate the vectors for each character of a word. This creates a new vectorial representation for a word coming from its singular characters, which (at least in European languages), often convey meaning on their own. For the aforementioned aggregation step, I tried two different models. The first was a CNN, which applied 2D convolutions to extract features from the character embedding matrix of each word, giving as output a single vector with 25 or 50 dimensions (*channels*, in convolution terms). The second was a BiLSTM, to analyse the sequential aspect of the characters: taking the output of this BiLSTM at the last timestep, we obtain a single vector that represents a word as a sequence of its characters. The output from the aggregator was then concatenated to the embedding vector of each word, and then fed to the main BiLSTM. Unfortunately, even after some hyperparameter tuning, none of this models reached more than 66-67% F1-score, showing severe overfitting. This probably means that the additional features were not meaningful enough to justify the added complexity of the model, and the new vectors were degrading the GloVe embeddings, which were already trained to perfection.

4.4 Addition of logic rules

The *BIO* tag system implies a set of rules that are not explicitly implemented in the model. For example, a sentence can never start with an I-tag, and an I-tag always comes after a tag of the same category. Even though these rules should be inferred by the model, it costs nothing to implement them, so I added code to detect labels that are surely wrong and infer new ones based on the previous and the next predicted label (figure 1). As expected, this improved the F1 score by a miniscule margin (in the order of 0.02%) proving that these patterns were already learned by the BiLSTM.

4.5 CRF

Lastly, as I had already verified that additions to the feature extractor worsened the model, I tried to improve the classifier, adding a *Conditional Random Field* (CRF) after the fully connected layers. In short, CRF (Lafferty et al., 2001) is a discriminative model that can be described as a Markov Random Field which is globally conditioned on a set of observations X. In the simplest (but most important) case, the graph is just a line, and it can be trained to perform context-aware sequence labelling. This aims at solving a crucial lack of our model: while the vectorial representation of each word is aware of its context, the classification step is isolated, meaning that the classification of a word is not influenced by that of its neighbours. The effects of this addition are very clear, as the validation F1-score improved to 75.97%.

5 Conclusions

To compare my results (summarized in figure 5) on a known dataset, I trained and tested my model on CoNLL-2003 (Sang and De Meulder, 2003a), which is much easier to classify, as it contains only 9 named entity labels, achieving a score of 91.22%. Figures (3, 4) show the cross-entropy loss and F1-score as they change over the epochs on both datasets: it is important to notice that, even after the loss starts increasing due to overfitting, the F1-score still has room for improvement. This is because the unbalanced cross-entropy loss does not take into account the skewness of the data, so a globally minimum loss does not lead to the best possible F1-score. Unfortunately, every strategy I tried to balance the loss, like *focal loss* (Sang and De Meulder, 2003b) or weighted cross-entropy, led to consistently worse results. The normalized confusion matrix in figure 2 shows how the only real problem of this model is the misclassification of named entities as *O*-tags, deriving from a bias of the English language reflected in the dataset.

$(i-1)^{th}$ tag	Wrong i^{th} tag	$(1+1)^{th}$ tag	New i^{th} tag	Notes
O	I1	I2	B2	
O	I1	O/B	B1	
B0	I1	O/B	B1	
B0	I1	I0	I0	never worse would be B0, but I0 is more reasonable
B0	I1	I2	B2	
I0	I1	O/B	B1	
I0	I1	I2	B2	
I0	I1	I0	I0	never worse would be B0, but I0 is more reasonable

Figure 1: Table of logic rules for the inference of the i^{th} tag based on previous and next ones.

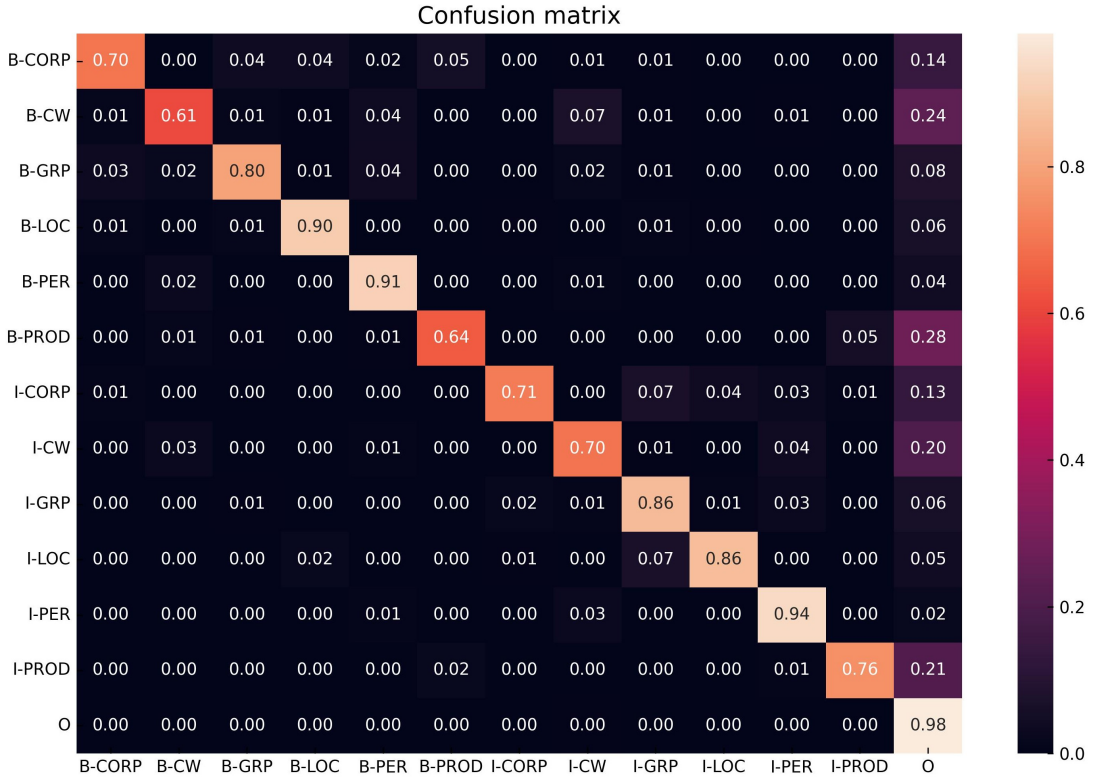

Figure 2: Normalized confusion matrix on the given dataset.

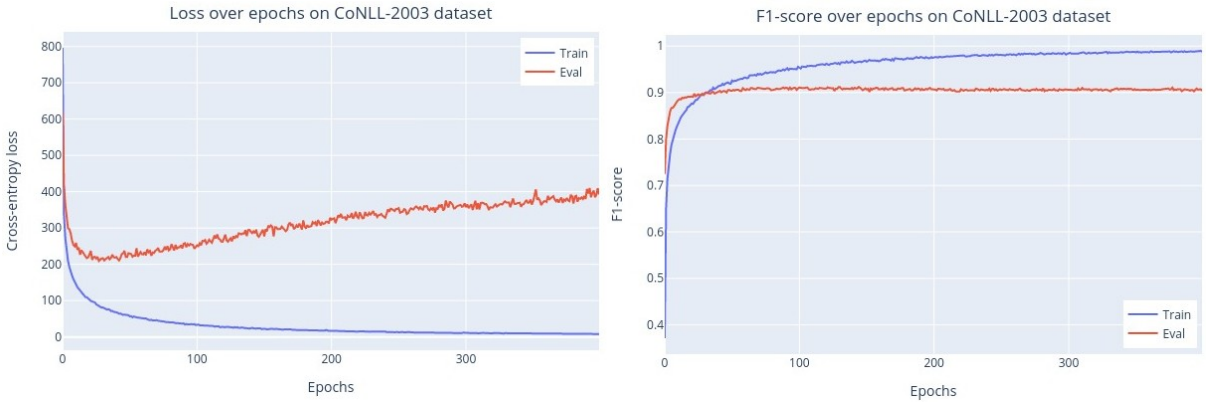

Figure 3: Loss and F1-score over epochs during training on CoNLL-2003 dataset.

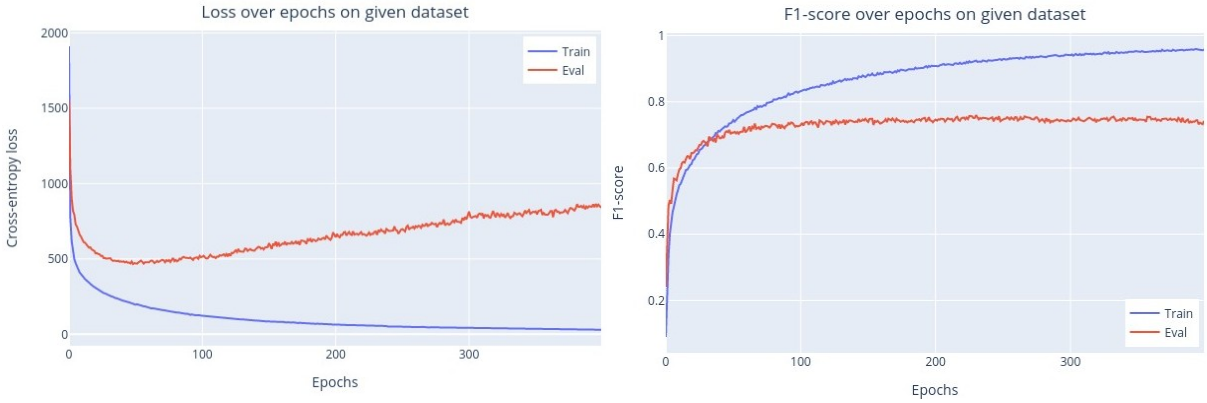

Figure 4: Loss and F1-score over epochs during training on the given dataset.

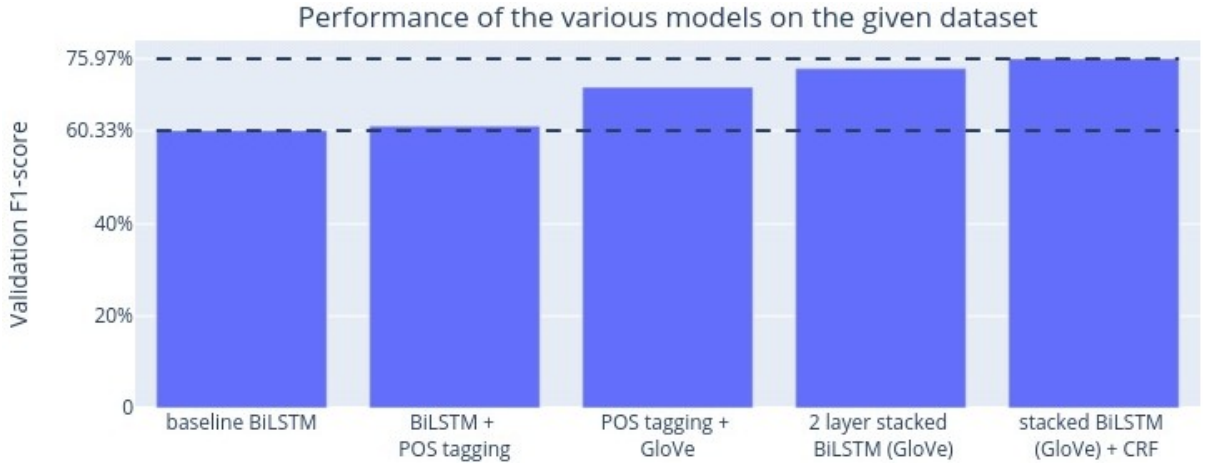

Figure 5: Comparison of the performance of different models on the given dataset

References

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Hiroki Nakayama. 2018. seqeval: A python framework for sequence labeling evaluation. Software available from https://github.com/chakki-works/seqeval.

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.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003a. Introduction to the conll-2003 shared task: Language-independent named entity recognition.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003b. Introduction to the conll-2003 shared task: Language-independent named entity recognition.