Named Entity Recognition

Ismagil Uzdenov
1873718
uzdenov.1873718@studenti.uniroma1.it

1 Introduction

Named entity recognition (NER) is the task of identifying the named entities in a given text. The named entities can be names of people, locations, organizations, dates, quantities and so on. The objective is to classify this entities into pre-defined categories. In this case these categories are `ORG` for organization, `PER` for person, `LOC` for location and `O` for other.

In this project we use a statistical approach using neural networks to classify each token in a given sentence with a NER tag.

2 Approach

The general approach is to use a *bidirectional long short-term memory* (BiLSTM) module to extract features from a given sentence and classify those using a *softmax layer* into categories. The BiLSTM takes word embeddings to avoid the sparsity problem that can occur from using one-hot encodings of the tokens.

In this project nine different setups are explored in which four different models and three different preprocessing setups are used. The models are as described below:

- `BiLSTM` : This model uses a word embedding layer, a BiLSTM layer and a softmax layer. It is used as a baseline for the other models.

- `BiLSTM_EXP` : For each instance, this experimental model conditions on the previous instance's output. This model was inspired by the BiLSTM-CRF model, as that model conditions an instance to the outputs of the whole sequence. The idea is to treat the setup as having the Markovian property, assuming that the previous instance contains all the needed contextual information that is needed to classify current instance, and condition on this information. See Figure 1.

- `BiLSTM_CHAR` : This model adds character embeddings to the baseline model.

- `BiLSTM_ULT` : This 'ultimate' model combines all the extra features in the other models. Namely, character embeddings, `Random UNK` setup (see Section 2.1) and experimental part of `BiLSTM_EXP`.

2.1 Preprocessing

As mentioned before, there are three preprocessing setups in this projects.

First setup, denoted as `True-case`, constructs the data set as it is given in the raw form, from capitalization point of view. In this setup, tokens are not lower-cased and are preprocessed in their original form.

Second setup, denoted as `Mixed-case`, constructs a data set containing both true-cased and lower-cased versions of the sentences. The idea is to construct a more robust model that can handle casing errors in the test set, in case there are any. This might be advantageous since in the NER task the casing of a token is an important sign that this token might indeed be a named entity, since names of people, locations, organization, etc. start with a capital letter. (1) suggests that this type of preprocessing greatly increases the accuracy of the NER model on the data sets containing instances of erroneous casing.

Third setup, denoted as `Random UNK`, randomly replaces tokens occurring only once in the data set with the 'UNK' tag. The aim is to train the 'UNK' tag in the vocabulary since the vocabulary is constructed from the training set and there will not be any 'UNK' tags during the training. Thus, embeddings with this tag will not be trained which might result in a poor result for that token

and in a bad effect on the context flowing through the BiLSTM layer. This process was done on the `True-case` data set.

Beside the above setups, the vocabularies are constructed from the training set for both word and characters. These vocabularies are the same for all experiments.

3 Experimental Setup

The experiments were made in the `Google Colab`. First, the vocabularies are constructed from the training set. Then using these vocabularies all the data sets are built, as it is described in Section 2.1. Everything from the processing of `.tsv` files to returning batches to the model is done by a single class named `DataHandler`, one for each file. This class handles the preprocessing, indexing the data, saving and loading the data sets and constructing batches. The indexing, denoted as *enumeration* in the code is done parallely on four processes to speed it up, since character indexing for each token takes a lot of time. Then, the models described in Section 2 are built.

For each experiment the corresponding setup is loaded and used in the training loop, which consist of several epochs. In each epoch the training data is traversed once and after that the validation data is used to calculate a reference loss without any update of the weights. After the training loop is over the resulting validation losses are compared to select the most successful model. The selected model is then used to classify the testing set and compute precision, F1 score and the confusion matrix.

In each experiment the same main architecture was used: an Embedding Layer with `dim=100`, a BiLSTM Layer with `dim=50` and a Softmax Output Layer with `dim=5`, one extra dimension for the padding instances. The `BiLSTM_CHAR` also uses an Embedding Layer with `dim=50` and a BiLSTM Layer with `dim=50` for character embeddings, and a *Dropout Layer* with `p=0.5` before BiLSTM, as it is described in (2), to ensure that both word and character embeddings are used in the classification. Learning rate was set to 0.001.

Due to time limitation, the time it takes to train a `BiLSTM_CHAR` model and apparent ineffectiveness (see Section 4) of `Mixed-case` setup, the experiment combining this model with this data set was not performed.

4 Experiment Results

The resulting F1 scores of the nine experiments can be seen in Table 1. The training and validation losses throughout the training loops can be seen in Figures 3-11. The confusion matrix for the best performing setup, which is `BiLSTM_ULT`, is shown in the Figure 2.

5 Analysis

The analysis between casing shows that `True-case` setup gives higher scores than the `Mixed-case` setup. This could be caused by the network having to optimize on the same context for two different tokens and the gradient jumping these instances. In theory this setup should work better than others on a data set with erroneous casing instances. Among the setups the most successful clearly seems to be the `Random UNK`. This makes sense since the 'UNK' tag is more optimized into the context flow in the network using this setup, because they were trained to classify those instances.

Looking at the models, one can clearly see that, although, simpler models perform good, the character embeddings have a profound effect on the results. This also makes, a lot of sense since NER as a task is very dependent on the semantics of the word, e.g., as mentioned before, casing. The character embeddings are able to capture this and therefore the resulting model outperforms previous ones.

The interesting thing is that `BiLSTM_ULT` setup outperforms all others. The `BiLSTM_EXP` model was underperforming with respect to the `BiLSTM` setup, yet the conditioning signal from previous instance combined with the character embeddings yields the best results.

The confusion matrix (Figure 2) shows that there is a big imbalance in the data set. Most of the NER tags belong to the `O` category. This makes it very easy to get a good result on a statistical model, since most of the gradient flow will affect this tags weights.

6 Conclusion

In conclusion, this project explores three models and three setups of named entity recognition task. While most of them provide good results, ultimately the combination of the models gives the best result.

References

[1] Stephen Mayhew and Tatiana Tsygankova and Dan Roth. *ner and pos when nothing is capitalized*. CoRR, 2019, 1903.11222.

[2] Guillaume Lample and Miguel Ballesteros and Sandeep Subramanian and Kazuya Kawakami and Chris Dyer *Neural Architectures for Named Entity Recognition*. CoRR, 2016, 1603.01360.

A Appendices

Module	True-case	Mixed-case	Random UNK
BiLSTM	0.8889	0.8795	0.8905
BiLSTM_EXP	0.8866	0.8795	0.8887
BiLSTM_CHAR	0.9235	N/A	0.9278
BiLSTM_ULT	N/A	N/A	**0.9295**

Table 1: The testing results of the experiments in terms of F1 scores.

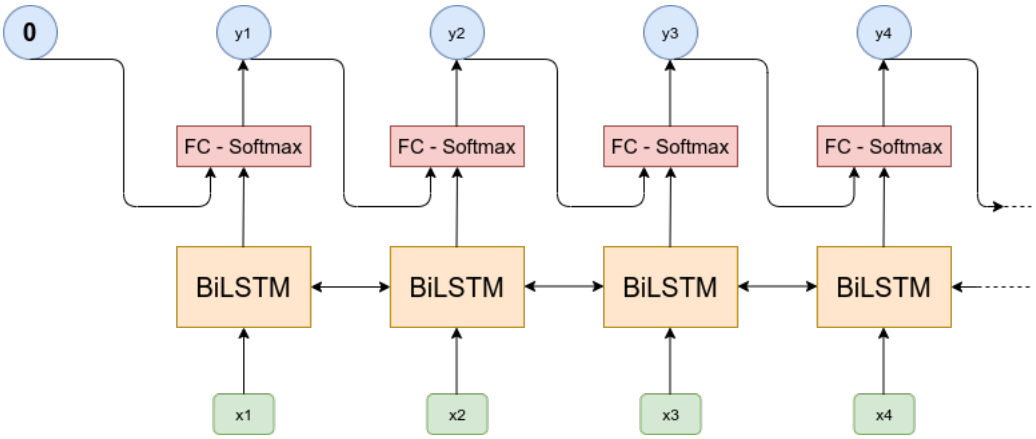

Figure 1: BiLSTM_EXP model overview.

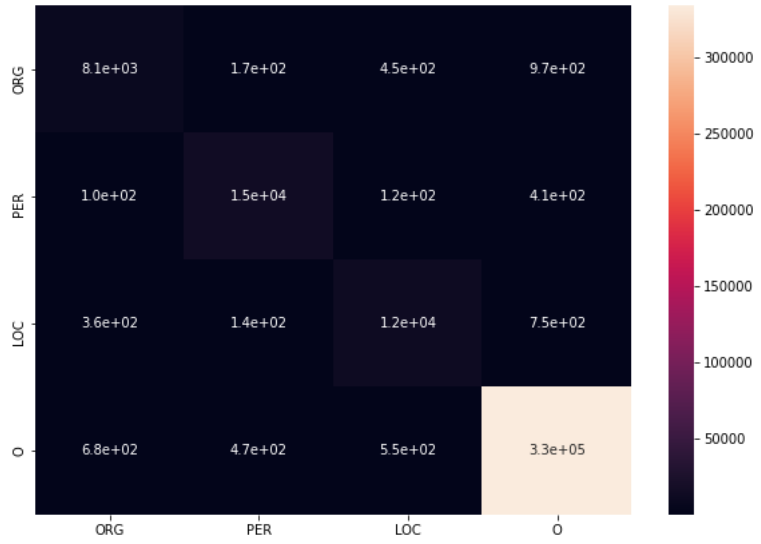

Figure 2: BiLSTM_CHAR, Random UNK: Confusion matrix.

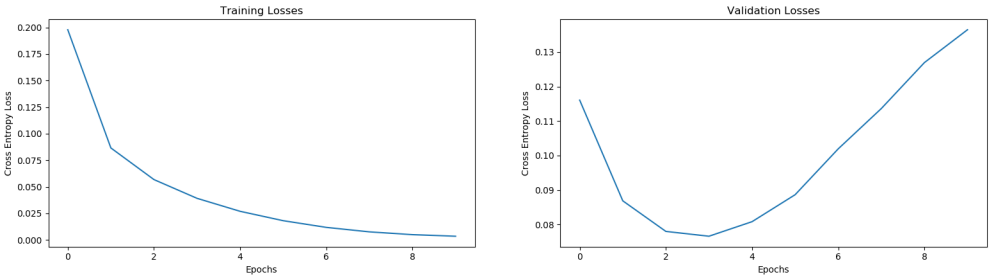

Figure 3: `BiLSTM`, `True-case`: training and validation losses over 10 epochs.

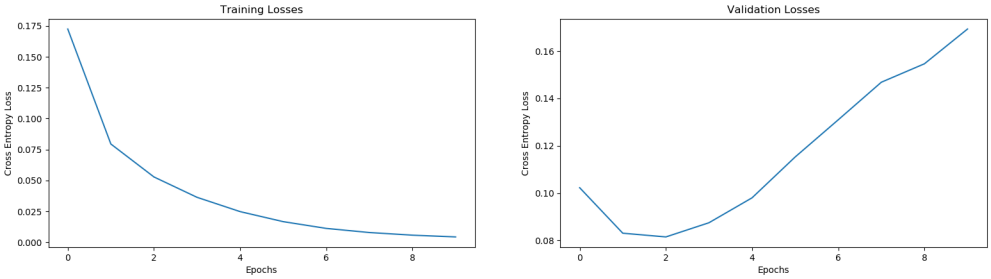

Figure 4: `BiLSTM`, `Mixed-case`: training and validation losses over 10 epochs.

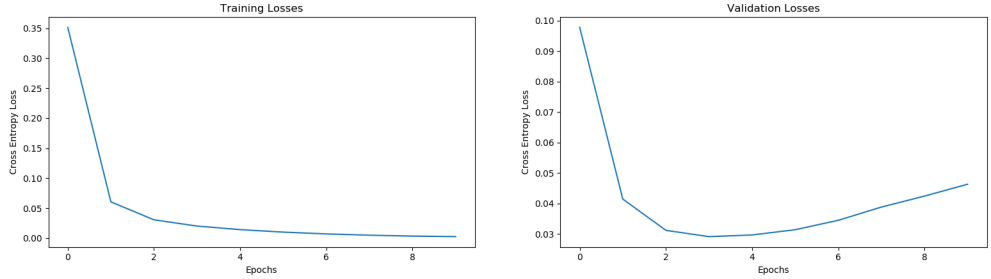

Figure 5: `BiLSTM`, `Random UNK`: training and validation losses over 10 epochs.

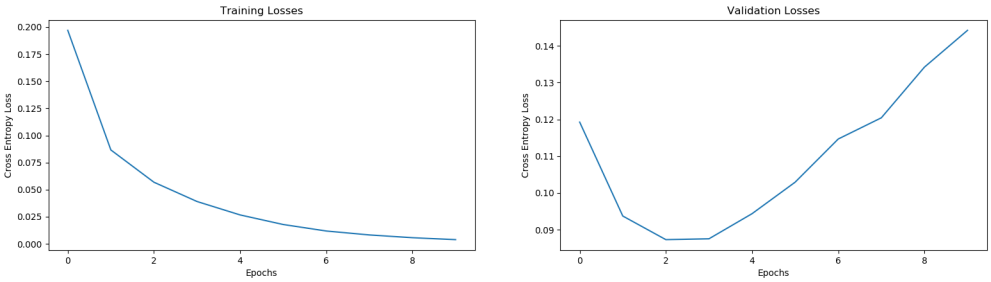

Figure 6: `BiLSTM_EXP`, `True-case`: training and validation losses over 10 epochs.

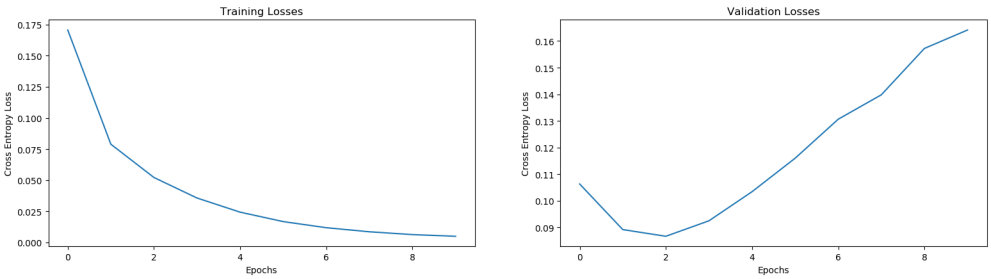

Figure 7: `BiLSTM_EXP`, `Mixed-case`: training and validation losses over 10 epochs.

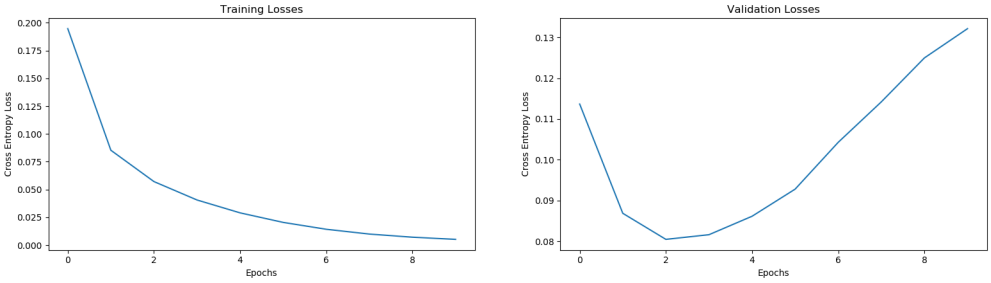

Figure 8: `BiLSTM_EXP`, `Random UNK`: training and validation losses over 10 epochs.

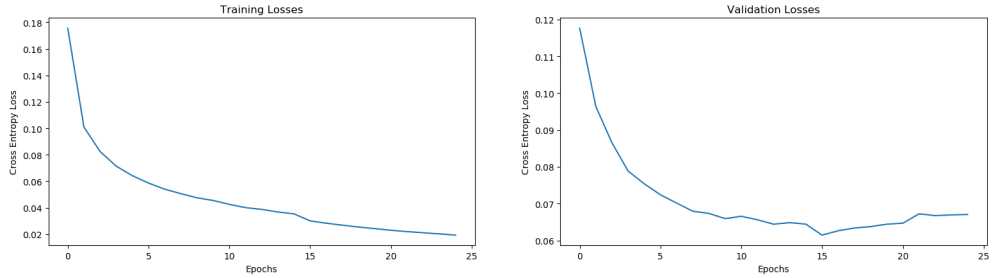

Figure 9: `BiLSTM_CHAR`, `True-case`: training and validation losses over 25 epochs.

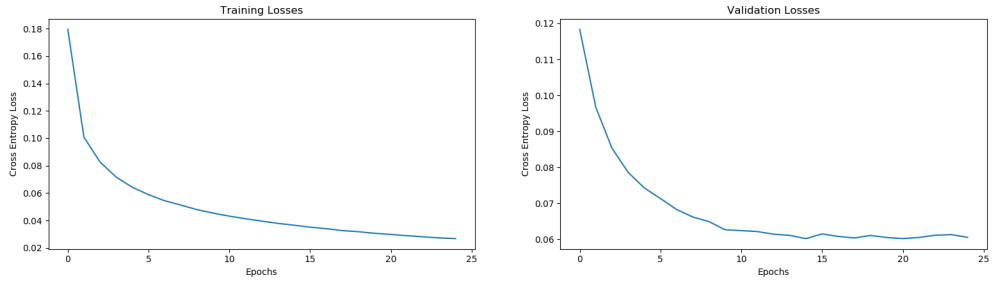

Figure 10: `BiLSTM_CHAR`, `Random UNK`: training and validation losses over 25 epochs.

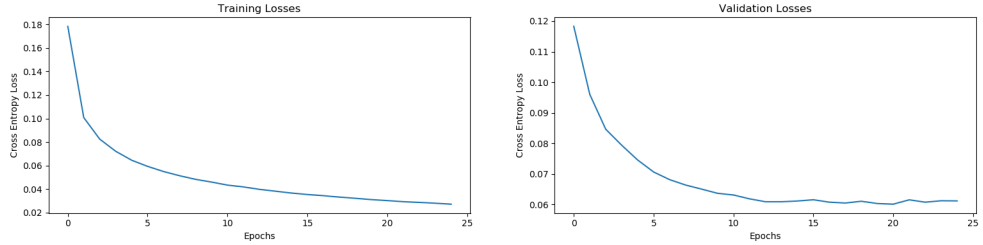

Figure 11: `BiLSTM_ULT`: training and validation losses over 25 epochs.