NLP Homework 1: Named Entity Recognition

Marco Lo Pinto

1 Introduction

In the field of Natural Language Processing, Named Entity Recognition is an information extraction task that seeks to classify each word in a text to a corresponding named entity label (such as "PER" for a person, "LOC" for a location and so on, with the special label "O" which means no entity). The tags that will be used in this assignment are in BIO format, where each named entity label has a prefix, which could either be "B-" or "I-", that determines if the tag is the beginning of a chunk or inside one (e.g. "New York is ..." is labelled as "B-LOC I-LOC O ..."). Therefore, some particular tags can never appear after others (like "... B-PER I-LOC ..."): a peculiar approach will be used in order to avoid these forbidden patterns.
The structure of the paper follows the chronological order of tests done in order to understand possible approaches and solutions.

2 Dataset Analysis

Before creating the general pipeline for this task, I analyzed the training (and dev) dataset in order to obtain some information about it and check if some preprocessing was needed. First, we can see form figure 1 that it is very unbalanced: there are more "O" tags than the sum of all the other ones; this means that simple metrics like accuracy could be misleading in order to check if the model predicts correctly. Secondly, both training and dev dataset don't contain capital letters, which means that it will be harder for the model to understand the type of entity for ambiguous words (e.g. Apple and apple). Moreover, there are symbols and characters that are not present in the English alphabet (like Chinese characters), which is something to keep in mind (even though there are very few phrases with them in it). With these considerations, no preprocessing was done (not even with numbers).

3 Model Architecture

3.1 Word Embedding Layer

Considering all the steps in order to create the network, after some tests I understood that choosing how and which word embedding to use is the most important one and can make a huge difference. The first word embedding that was tried is FastText (Bojanowski et al., 2016): the reason behind it is to remove the hassle of dealing with OOV words. This implementation has a small vocabulary (compared to the other tests) but with lots of n-grams. A simple model with this embedding, a bidirectional LSTM and a linear layer as classifier performed very badly and almost saturated the RAM of my machine, with an f1-score of $\sim$38%. Thus, a different approach was tried in the second test: a GloVe (Pennington et al., 2014) embedding with a larger vocabulary (400'000 words, excluding <pad> and <unk> special tokens for padding and OOV words), giving a boost in performance and arriving at $\sim$57%. This means that a bigger vocabulary is better than a smaller one but with lots of n-grams. Other word embeddings were tried, like a pretrained Word2Vec with a massive size of 3'000'000 vectors, but it was immediately discarded for two reasons: it (surprisingly) had less known words for the training set and also it required an amount of RAM that was not feasible for my machine.
Another test done was to see the differences between a model with a freezed word embedding layer and the same model but with the possibility to modify its weights. As we can see from figure 3, performances were better with a freezed embedding layer.

3.2 Classifiers

Last layers of the model are the ones that uses features extracted from previous ones in order to get

most probable labels. For this part, at the beginning a simple linear layer was used, reaching performances up to ∼57%. Thus, a different approach was tried, adding to the last layer a (linear-chain) Conditional Random Field layer (Lafferty et al., 2001): its transition matrix generates a sequence likelihood where there is the dependency between each state (the labels) and the entire input sequence. This is important to create some forbidden transitions in the matrix, as shown in figure 4. First tests were made by handcrafting the transition matrix (figure 5) and then fine-tuning it with training, reaching ∼62% of f1-score. In another test (figure 4), the matrix was initialized with random values and then the model was trained, reaching ∼69% of f1-score. We can see from the two figures that some intuitions made on the handcrafted matrix were right.

3.3 LSTM

The starting model consisted of a bidirectional LSTM with two recurrent layers: this is because a simple LSTM is not enough for this particular task. Adding another layer made the simple model improve performances, independently of the embeddings and classification layers; thus the final LSTM uses three recurrent layers.

3.4 Part-of-Speech integration

PoS tagging is a process that categorize words in a sentence depending not only on the definition of the word, but also its context. This means that these generated information could be useful for NER tasks, because proper nouns are very likely to be named entities, whereas punctuations and verbs are not. With that in mind, in the fourth test uses the features extracted from the output of the LSTM and combines it with the one-hot-encoded vector of the PoS tag. Unfortunately, the final result didn't improve the performances of the model, so the PoS part was removed from the final implementation.

3.5 Character-level integration

Another way tested was to implement character-level embedding (Ling et al., 2015): this was done by generating a vocabulary from characters contained both on GloVe dictionary and the training set: there were in total 577 different characters, from English to Chinese symbols. The character embedding layer was implemented into the model and its weights were unfreezed, combined with a many to one LSTM layer. This strategy pushed a little bit higher the f1-score, reaching ∼72%. Even if this change is very small, it's important to remember that, as already stated, there are words that are not present in the GloVe embedding and this implementation can take care of OOV words better compared to the UNK token.

4 External data for training

Sometimes if the data provided for the training part is not sufficient, then the model could easily overfit; this is because the fewer samples, the more models can fit the data. In order to understand if that's the case, since for this homework the use of external data is allowed, in the sixth test the training dataset was combined with the English dataset used in the WikiNEuRal paper (Tedeschi et al., 2021). The training dataset went from ∼14'000 sentences to ∼58'000. After some training done with the final model (figure 8), there wasn't an improvement in performances, meaning that the data already available was sufficient for this task.

5 Hyperparameter tuning

In order to do improve the model, hyperparameter tuning can be helpful. This was done by searching the best optimizer (Adam, SGD or RMSProp) with the best parameters (such as momentum, weight decay and so on) and the best parameters for the model (such as number of layer for the LSTMs, dropout and so on). Unfortunately, the improvement was marginal (∼1% for f1-score) because most of the tests done performed equally or worse than the original implementation.

6 Results and Conclusions

The final matrix of the model is reported in figure 7 and shows that it has a tendency to predict as entities words that are labeled as "O".
The model performed differently by changing different word embeddings, meaning that by using more sophisticated ones (like contextualized word embeddings such as ELMo) the f1-score would certainly increase. My hypothesis is supported by the fact that the performance of the simple initial model (composed of word embedding, LSTM and linear layer for classification) gave a similar result to the final model. In the end, what really made the difference was the choice of word embedding and also the addition of a (linear-chain) Conditional Random Field in the last layer of the model.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramón Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal. Association for Computational Linguistics.

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.

Simone Tedeschi, Valentino Maiorca, Niccolò Campolungo, Francesco Cecconi, and Roberto Navigli. 2021. WikiNEuRal: Combined neural and knowledge-based silver data creation for multilingual NER. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2521–2533, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Model	F1-score
Bi-LSTM + FastText	0.3764
Bi-LSTM + GloVe	0.5689
Bi-LSTM + GloVe + CRF (by hand)	0.6140
Bi-LSTM + GloVe + CRF	0.6897
Bi-LSTM + GloVe + CRF + Char	0.7302

Table 1: Summary of architectures and their f1 scores on the dev set.

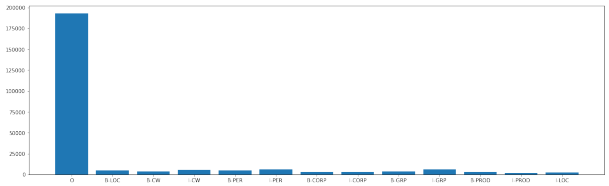

Figure 1: Distribution of the tags from the training dataset.

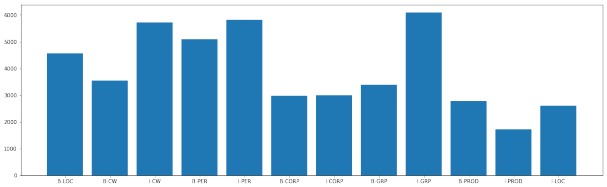

Figure 2: Distribution of the tags from the training dataset (without the "O" tag).

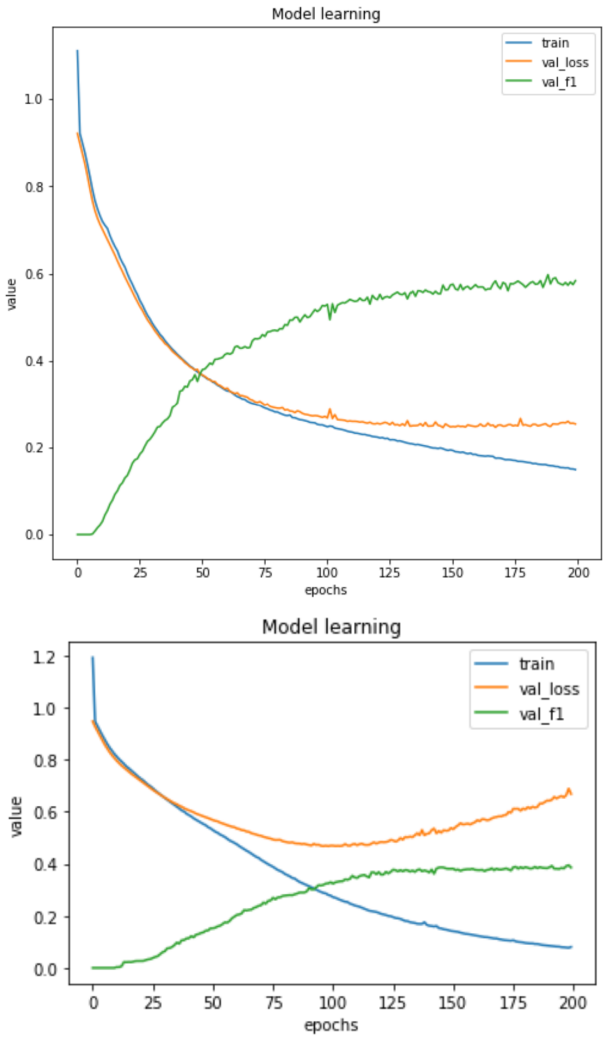

Figure 3: The same model but with freezed pretrained word embedding (top) and left unfreezed in order to train it (bottom).

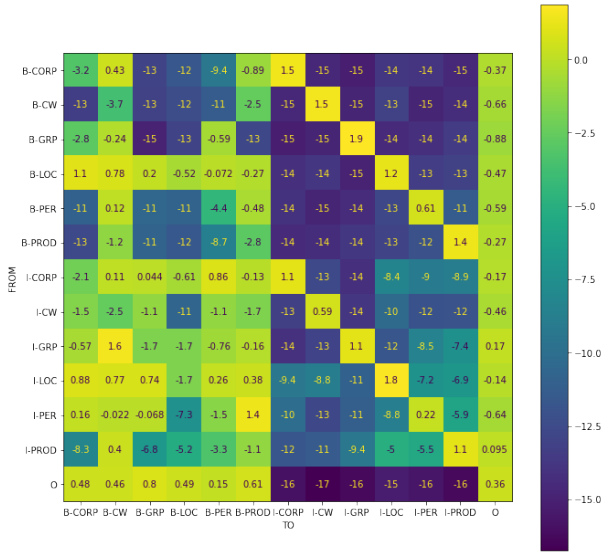

Figure 4: The transition matrix of the CRF layer. It can be seen that the model is learning the rules: from ”O” tag to ”I-” shouldn’t happen, as well as from a ”B-” tag to a different entity with ”I-” tag.

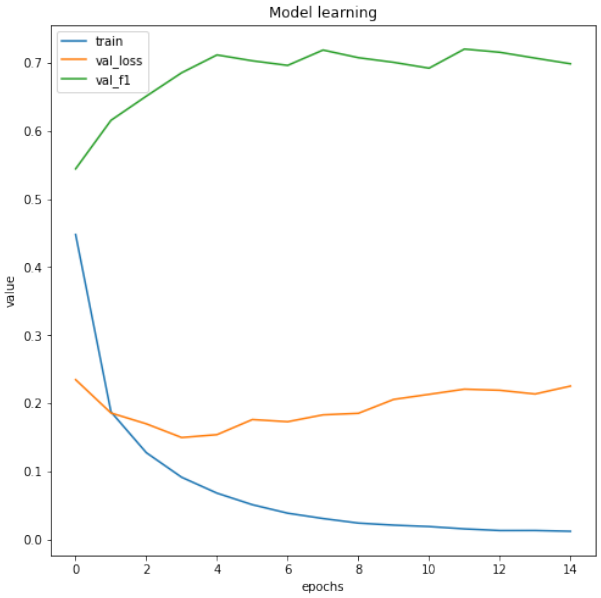

Figure 6: The training history of the final model. Weights were saved before overfitting.

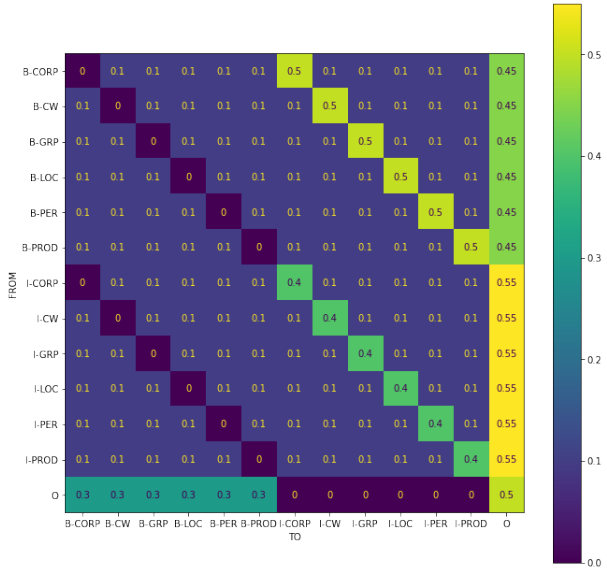

Figure 5: The handmade transition matrix of the CRF layer. It can be seen that it’s very similar to the one that the model learned.

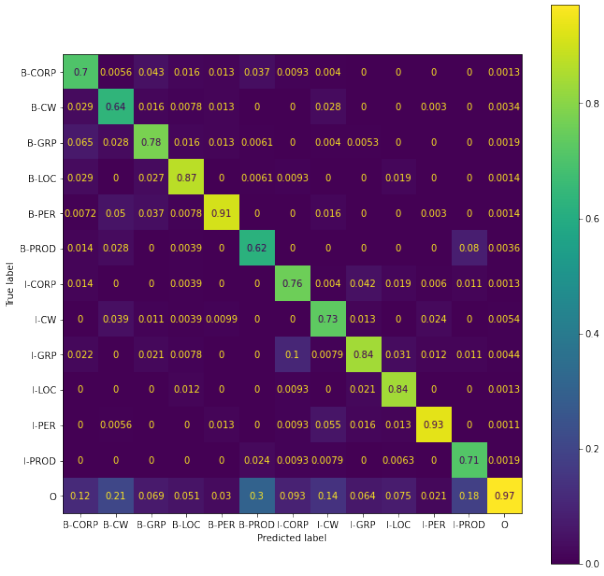

Figure 7: The confusion matrix of the final model normalized over predictions.

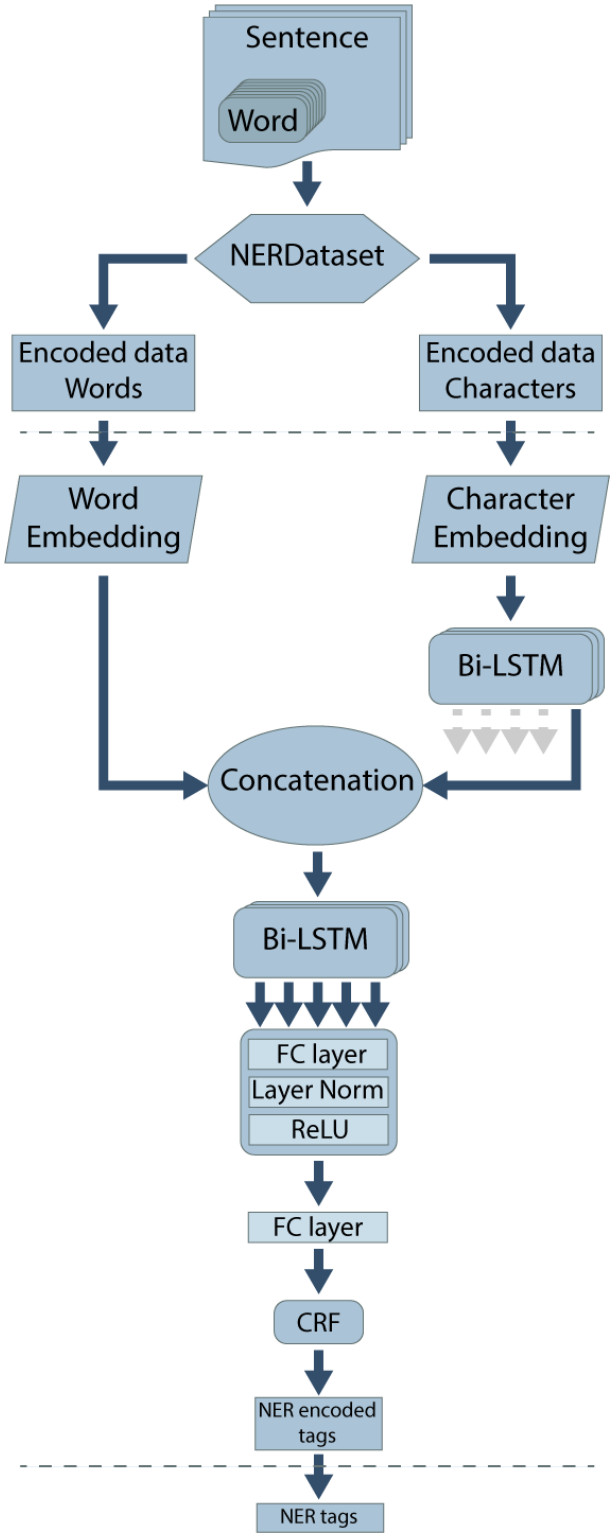

Figure 8: The final model architecture. The model receives the encoded words and characters as input and generates NER encoded as integers. Then, using the label vocabulary, NER integers can be converted to strings.