NLP homework 1: Named Entity Recognition

Anonymous ACL-IJCNLP submission

1 Introduction

Named Entity Recognition is the task of automatically identifying named entities in a text and classifying them into some given categories, which depend on the chosen dataset. In our case we have 7 classes: PER, LOC, GRP, CORP, PROD, CW and O for no named entity. My work started by implementing a simple LSTM baseline model, then I started adding one new component at a time on top of the model and only if it brought improvements on the validation set then it was kept, the sections of this report follow the chronological order of the extensions that I added. The final model is a BiLSTM-CRF that uses the input informations of pretrained word embeddings and POS tags embeddings, which has already been proved to be effective for this kind of problem (Panchendrarajan and Amaresan, 2018; Lample et al., 2016).

2 Data insights

As any other machine learning task, I started from looking at the data and I noticed some interesting patterns. First of all the datasets are all lowercase, this is important because it makes the learning process harder, since capital letters are key features in the identification of named entities. Moreover there are not only English words and Latin characters (e.g. Arabic words are present), but since these are very few they will not interfere with the use of pretrained English word embeddings.

3 Model architecture

3.1 Baseline

Given an input sentence as a sequence of words, each word needs to be transformed into a meaningful numerical representation, hence an embedding layer is used for this purpose, mapping each word into an higher-dimensional latent space. The vocabulary considered to build the embedding layer is the full set of words present in the dataset, this choice was made first of all because there are not many word types (29965) and then because this vocabulary performed better than filtering words with a minimum frequency. After the embedding layer this model has a single layer LSTM (Hochreiter and Schmidhuber, 1997). Long Short Term Memory (LSTMs) are a type of recurrent neural networks that partially solve the vanishing gradient problem by introducing a cell state in addition to the hidden state, hence they can learn long-term relationships in input sequences of variable length and for this reason they are tremendously effective in tasks like NER. At the end, a linear layer with Softmax activation function is used to classify each token. This simple model reaches an F1 score of **47.4%**, its confusion matrix is in Image 2.

3.2 Bidirectional and Dropout

The main drawback of LSTMs is the fact that they process the input forward from left to right, so for each token the network only knows the context before it. This is not ideal because in order to correctly identify a named entity also the context after it is important, as there may be some references to the entity that are key factors in recognizing it (e.g. in *"Yes, Stanley Kubrick directed the movie."* knowing the context after the named entity is essential). Bidirectional LSTMs (Graves and Schmidhuber, 2005) solve this problem by stacking two LSTMs on top of each other: one will process the input sequence in forward order encoding the context before every token, and one will process it in reverse order encoding the context after each token. The two hidden representations obtained for each token are then concatenated in output. Moreover, many LSTMs or BiLSTMs can be stacked to form deep architectures, allowing the network to capture higher-level features. For all these reasons I switched to a 3-layers BiLSTM that gave me an

F1 score of **60.2%**. Then I also decided to add Dropout after each layer because the number of parameters increased and I wanted to avoid overfitting. Dropout (Hinton et al., 2012) is a regularization technique that randomly zeroes units of the neural network during training and this led me to **62.3%** F1.

3.3 Pretrained word embeddings

Next, I switched from randomly initialized word embeddings to pretrained word embeddings, which better capture the semantic and syntactic relationships between words because they are trained on larger datasets. I tried both Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) embeddings exposed through Gensim (Řehůřek and Sojka, 2010), with and without fine-tuning. The results can be found in Table 1. It's interesting to notice that fine-tuning on Word2Vec vectors increased their initial performance, meanwhile fine-tuning on GloVe vectors decreased it. A possible explanation may be due to the fact that Word2Vec embeddings are trained on the Google News dataset, hence they are more generic, whereas GloVe ones are also trained on Wikipedia, which contains a lot of pages for named entities, so they are already specialized. Anyway, the best performance came from the non-fine-tuned GloVe 300-dimensional embeddings, which achieved a **66.7%** F1 score on validation set.

3.4 Part of speech tags

Part Of Speech (POS) Tagging is a main task in NLP that can often improve many downstream tasks. Adding POS tags to the input tokens is useful because of the intuition that some words are most likely to be named entities with respect to other ones (e.g adverbs and interjections alone are very rarely named entities, while proper nouns are very often). To assign a POS tag to every word token I used the pretrained Stanza POS tagger (Qi et al., 2020), then every POS was mapped into an embedding and concatenated to the respective word embedding. The resulting vector is fed to the BiLSTM. This choice was made to avoid building a dedicated LSTM for POS sequence processing, therefore keeping the parameters number low. Also, Universal POS tags were chosen over Penn Treebank ones for the same latter reason. With this addition, the model reached an F1 score of **69.2%**. At this point I also tried adding more layers to the linear part after the BiLSTM because I thought the model would benefit from this, but it did not.

3.5 Conditional Random Field

So far, the i-th prediction of the last linear layer doesn't take into account what was predicted at position $i-1$, i.e. the classification sequence is composed of independent decisions. This behavior is not realistic, because predictions should actually be influenced by previous ones, in particular we would like to impose some constraints like: the first tag in a sequence can only be "B-" or "O", "B-x I-y" is a valid pattern only if x and y are the same label, and so on. Conditional Random Fields (Lafferty et al., 2001) are a special case of Markov Random Fields that can learn this kind of constraints. They combine the emission scores (logits) for each token, coming from the linear layer, with the transition scores (the actual parameters of the CRF, representing the "likelihood" we assign to the transition from label x to label y in a sequence) to predict the most likely sequence of labels. A CRF added on top of the BiLSTM forces the model to focus on producing correct sequences of labels rather than correctly classifying individual tokens. This completed my model, reaching a **70.9%** F1 score on validation. Its confusion matrix is in Image 3.

4 Training

After an extensive random search over hyperparameters as reported in Table 3, the final model is trained using Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.001 and an L2 regularization of 1e-5, to reduce overfitting. For this same reason, every experimented model is trained for 60 epochs using early stopping with 9 epochs of patience, thus causing a model to rarely be trained for more than about 40 epochs. Moreover, the saved weights are the one that produced the best validation F1 score during the training, this avoids saving noisy weights and further reduces overfitting, as you can see in Image 1. Batch size was fixed to 32, and the models were trained on a local GPU. When not using CRF the loss to be minimized is the Cross Entropy, otherwise Negative Log Likelihood is used and it is automatically computed by the CRF layer, thanks to the TorchCRF library [1]. Finally, Table 2 reports a summary of the improvements obtained by adding components to the model, whereas in Table 4 you can find a comprehensive classification report of the final model.

[1] https://github.com/s14t284/TorchCRF

References

A. Graves and J. Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4.

Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint*, arXiv.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

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

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013.

Rrubaa Panchendrarajan and Aravindh Amaresan. 2018. Bidirectional LSTM-CRF for named entity recognition. In *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*, Hong Kong. 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.

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*.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

Embedding	Frozen	Fine-tuned
GloVe 50	60.8%	•
GloVe 100	66.0%	•
GloVe 200	66.2%	•
GloVe 300	**66.7%**	63.2%
Word2Vec 300	56.5%	62.9%

Table 1: Comparison of F1 scores on validation set obtained using different pretrained embeddings. I decided to fine-tune and compare only the best GloVe embeddings with Word2Vec ones.

Component	Validation F1 score
Baseline	47.4%
+ bidirectional	60.2%
+ Dropout	62.3%
+ GloVe embeddings	66.7%
+ POS tags	69.2%
+ CRF	**70.9%**

Table 2: Summary of the added components with the respective improvement they produced. Notice that F1 score is taken as the only metrics because accuracy is not suitable for NER, due to the high class imbalance towards the O tag.

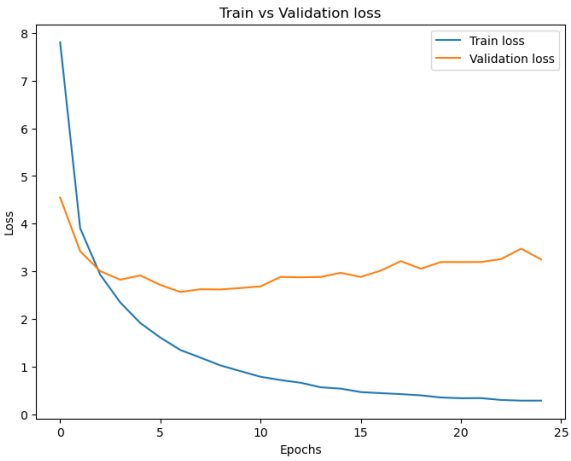

Figure 1: Train vs validation loss of the final model. The best performing weights were saved at epoch 16, patience was consumed at epoch 24.

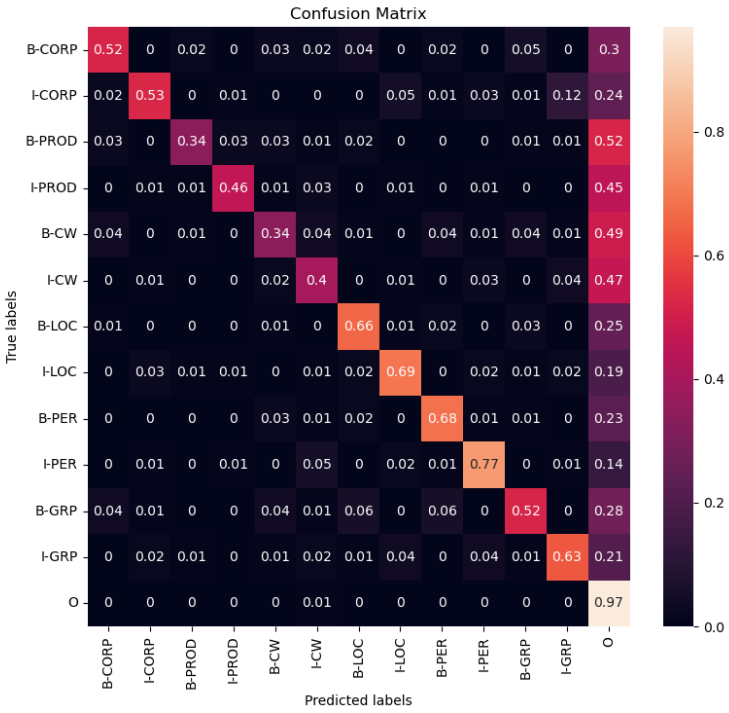

Figure 2: Token-based confusion matrix of the baseline model. We can notice that the O class is already almost perfectly classified because it is obviously the majority class. PER class seems the easiest to recognize, meanwhile PROD and CW are the harder ones because they are the minority classes.

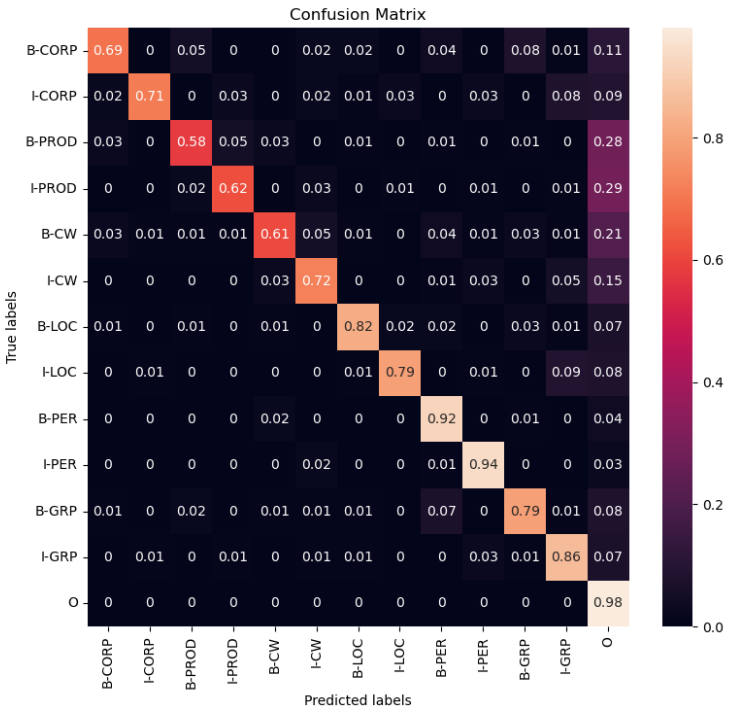

Figure 3: Token-based confusion matrix of the final model, we can see that every class improved as compared to the baseline model and the patterns are still the same. Also, the O tag is widely less predicted.

Hyperparameter	Values
Bidirectional LSTM	**True** \False
CRF	**True** \False
Dropout	0.2 \0.3 **0.35** \0.4 \0.5
LSTM's hidden dim	128 **256**
Word embeddings size	50 \100 \200 **300**
POS embeddings size	10 **25** \50 \100
LSTM layers	1 \2 **3** \4
Linear layers	**1** \2 \3 \4
Learning rate	1e-4 \5e-4 **1e-3** \5e-3 \1e-2
L2 regularization	0 \1e-6 **1e-5** \1e-4 \1e-3 \1e-2

Table 3: Random search was performed on these hyperparameters. Final model's hyperparameters are highlighted in bold.

	Precision	Recall	F1 score	Support
CORP	0.72	0.64	0.68	133
CW	0.60	0.55	0.57	170
GRP	0.73	0.75	0.74	190
LOC	0.84	0.80	0.82	243
PER	0.83	0.91	0.87	300
PROD	0.54	0.54	0.54	149
micro avg	0.74	0.73	0.74	1185
macro avg	0.71	0.70	0.70	1185
weighted avg	0.73	0.73	0.73	1185

Table 4: Classification report of the final model.