Named-entity recognition

Analysis of the effectiveness of adding a CRF layer on top of a LSTM network for Named Entity Recognition.

Author: Nico Manthey

August 30, 2020

1 Introduction

Named Entity Recognition (NER) is an important Natural Language Processing (NLP) task and a considerable amount of research is spend on it. Given some input text, the task is to mark named entities, commonly used entity categories encompass:

- states
- cities
- named structures (e.g. world trade center)
- geographical landmarks (rivers, mountains, seas)
- persons of public interest
- historical events
- companies
- NGOs
- institutions (e.g. the supreme court, the white house)
- currencies

... everything that can be referred to a name.

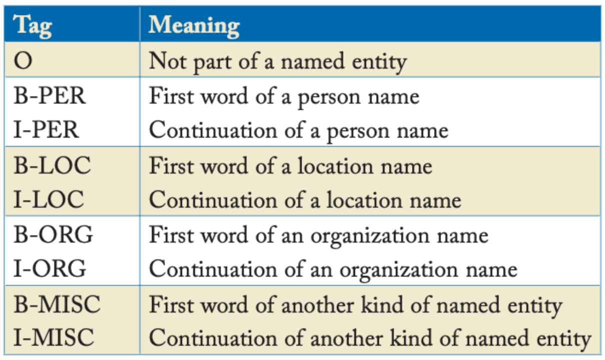
However, Named Entities are not uniquely identifiable by their names which makes the task to identify a NE context-dependent. Consider the following two sentences: "The U.S. sanctions eleven Chinese companies over human rights abuses in Xinjiang". In this sentence "U.S." refers to government, e.g. an organisation or institution, as opposed to the sentence: "The Yosemite National Park is considered one of the most beautiful national parks in the U.S.", where "U.S." refers to the country, e.g. a geographic region. Due to this context-dependent ambiguity the task of NER is not of deterministic nature and not solvable (or only very badly solvable) using lookup tables but probabilistic instead.

The NER task is normally modelled as a sequence tagging task. Text which is to be tagged gets pre-processed, which normally means all letters are converted to lower-case and the text is chunked into tokens where special characters are treated as tokens of its own. The so-obtained list is fed to a NER-model and the output is a list of the same length as the input sequence, where each token is assigned a tag. For the decision whether a word or token belongs to a Named Entity or not a model normally uses contextual information from all words in the sentence.

Two models which are commonly used for this task are Recurrent Neural Networks (RNN)[Elman 1990], especially Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber 1997] networks and Conditional Random Fields (CRF) [Ling et al. 2015], the use of which for NER has been analysed in [Lample et al. 2016]. Inspired by that paper, the aim of this work is to study the effectiveness of a CRF layer put on top of a LSTM network for Named Entity recognition. For this purpose, a

LSTM network similar but not identical to the one in [Lample et al. 2016] will be trained on the english CoNLL-2003 dataset. Additionally, a second network, identical to the first network with the only alteration that a CRF layer is inserted after the LSTM layer, will be trained on the same dataset. Finally, the two moels will be evaluated on the test set and their performance will be evaluated. This evaluation will allow to make some statements about the effects of adding a CRF layer to a network for Named Entity Recognition and sequence tagging. The implementation can be found on github [1]

A tagging scheme which is often used is called BIO. BIO is an abbreviation for "Begin, Inside, Outside". Since named entities can span over multiple tokens, the first token of a Named Entity is always marked with a B-tag and the following tokens that belong to the same entity are marked with and I-tag which is nicely illustrated in figure 1.



Tag	Meaning
O	Not part of a named entity
B-PER	First word of a person name
I-PER	Continuation of a person name
B-LOC	First word of a location name
I-LOC	Continuation of a location name
B-ORG	First word of an organization name
I-ORG	Continuation of an organization name
B-MISC	First word of another kind of named entity
I-MISC	Continuation of another kind of named entity

Figure 1: BIO tags as described in [Goldberg 2017, Table 7.1, page 81]

If you applied the BIO-tag set illustrated in figure 1 to the sentence "Virgina Woolf delivered the lectures at Newnham College, Cambrige, in October 1928..", this sentence would be converted to [("Virgina", B-PER), ("Woolf", I-PER), ("delivered", O), ("the", O), ("lectures", O), ("at", O), ("Newnham", B-ORG), ("College", I-PER), (",", O), ("Cambrige", O), (",", O), ("in", O), ("October", O), ("1928",O), (".", O)]. Note that commas and the dot at the end of the sentence are tokens of their own.

NER is a crucial component for further downstream tasks like dialogue systems, information extraction, question answering. In dialogue systems a flight booking agent uses NER to extract the origin and destination in a user query [**ATIS**]. In Information Extraction (IE) NER is important to recognize entities in unstructured text such that information about them can be extracted and further processed. Named entities can be more than the categories listed above. In amazon's alexa, for example, a named entity can also be any name of a smarthome device that a user might possess.

2 LSTM Model and LSTM-CRF Model

2.1 LSTM Model

Recurrent Neural Networks (RNNs) are designed to work on sequential data and the Long-Short Term Memory (LSTM) network [Hochreiter and Schmidhuber 1997] is an improved version of the classic Elman RNN [Elman 1990]. The input to an LSTM is an ordered sequence of vectors of arbitrary length $(x_1, x_2, x_3, ..., x_n), x_i \in \mathbb{R}^{d_{in}}$ where n is the length of the sequence and d_{in} is the fixed dimensionality of the ith input vector. The output is an ordered sequence of vectors

[1] https://github.com/nicoManthey/DL4NLP_final_project_NER_MANTHEY

$(h_1, h_2, h_3, ..., h_n), h_i \in \mathbb{R}^{d_{out}}$ of the same length, where d_{out} is the fixed dimensionality of the ith output vector. RNNs were designed to overcome the Markov assumption that a word in a sequence is only dependent on the previous word. Instead, the output of an RNN at every time step is conditioned on the entire previous sequence. In addition, LSTMs were designed to overcome the problem that weight updates were propagated poorly to the early layers of the RNN which resulted in poor learning behaviour when dealing with long sequences. This is also known as the vanishing gradient problem. To address the vanishing gradient problem the LSTM cells incorporate a memory and a forget gate.

$$\boldsymbol{s_t} = \boldsymbol{R}_{LSTM}(\boldsymbol{s_{t-1}}, \boldsymbol{x_t}) = [\boldsymbol{c_t}; \boldsymbol{h_t}] \tag{1}$$
$$\boldsymbol{c_t} = \boldsymbol{f} \odot \boldsymbol{c_{t-1}} + \boldsymbol{i} \odot \boldsymbol{z} \tag{2}$$
$$\boldsymbol{h_t} = \boldsymbol{o} \odot \tanh(\boldsymbol{c_t}) \tag{3}$$
$$\boldsymbol{i} = \sigma(\boldsymbol{x_t W^{xi}} + \boldsymbol{h_{t-1} W^{hi}}) \tag{4}$$
$$\boldsymbol{f} = \sigma(\boldsymbol{x_t W^{xf}} + \boldsymbol{h_{t-1} W^{hf}}) \tag{5}$$
$$\boldsymbol{o} = \sigma(\boldsymbol{x_t W^{xo}} + \boldsymbol{h_{t-1} W^{ho}}) \tag{6}$$
$$\boldsymbol{z} = \tanh(\boldsymbol{x_t W^{xz}} + \boldsymbol{h_{t-1} W^{hz}}) \tag{7}$$

Figure 2: Description of LSTM components, inspired by [Goldberg 2017, formula 15.4, p.180]

Where $\boldsymbol{s_j} \in \mathbb{R}^{2*d_h}, \boldsymbol{x_j}\ in \mathbb{R}^{d_x}, \boldsymbol{c_j}, \boldsymbol{h_j}, \boldsymbol{i}, \boldsymbol{f}, \boldsymbol{o}, \boldsymbol{z} \in \mathbb{R}^{d_h}, \boldsymbol{W^{xo}} \in \mathbb{R}^{d_x x d_h}, \boldsymbol{W^{h0}} \in \mathbb{R}^{d_h x d_h}$. Also, σ is the sigmoid function and $\odot$ is the element-wise product. In the formulas above $\boldsymbol{s_t}$ is the current state of the network at time step t and it is yielded by applying the LSTM cell to the state of the previous time step $\boldsymbol{s_{t-1}}$ and the input token of the current time step $\boldsymbol{x_t}$. The state is the concatenation of the memory component $\boldsymbol{c_t}$ and the hidden state $\boldsymbol{h_t}$. The output of the LSTM in every time step is h_t. The output of the LSTM for the entire sequence is $h_{0..t}$.

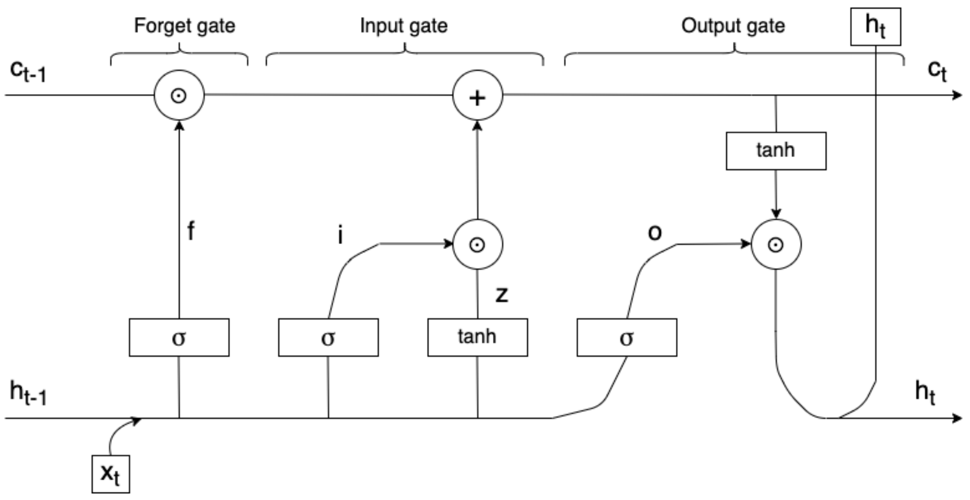


Figure 3: Illustration of an LSTM cell.

The LSTM model in [Lample et al. 2016] uses a bi-directional LSTM. A bi-directional LSTM (bi-LSTM) is formed of two LSTM networks, one reading the input sequence from the left side and one reading the input sequence from the right side. They both yield a hidden state for every time step. The hidden states of the entire sequence are obtained by concatenating the two hidden states $\boldsymbol{h_{0..t}} = [\overrightarrow{\boldsymbol{h_{0..t}}}; \overleftarrow{\boldsymbol{h_{0..t}}}]$. While in normal LSTMs the hidden state at time step t only captures information from all previous tokens in the sentence the bi-LSTM captures information from all tokens that appear before and after the current token $\boldsymbol{x_t}$ and is therefore better suited for the NER sequence tagging tasks.

The output $\boldsymbol{h_{0..t}}$ of the bi-LSTM is connected to a fully-connected layer to transform the bi-LSTM output such that it gives a proability distribution over BIO tags. Finally, the model output, e.g. the predicted BIO tags $\boldsymbol{\hat{y}}$ are obtained by taking the softmax.

2.2 LSTM-CRF Model

The LSTM-CRF model is similar to the LSTM model with the only difference that a Conditional Random Field (CRF)) [Ling et al. 2015] is inserted after the fully-connected layer. A CRF models the conditional probability $p(\boldsymbol{y}|\boldsymbol{X})$ where, in our case, $\boldsymbol{y}$ is a vector representing a sequence of BIO tags and $\boldsymbol{x}$ is a vector representing the input sentence. The LSTM-CRF model is equal to the LSTM-CRF model used in [Lample et al. 2016]. The input to the LSTM is a matrix $P \in \mathbb{R}^{nxk}$ where n is the length of the sequence and k is the number of tags. $P_{i,j}$ is the score of tag j for the word i for a given sentence. A sequence of predictions $y = y_1, y_2, ..., y_n$ has a score:

$$s(\boldsymbol{X}, \boldsymbol{y}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_1} \tag{8}$$

$\boldsymbol{A}$ represents a matrix of transition scores and $A_{i,j}$ is the transition score from tag i to tag j. Taking the softmax over all possible tag sequences gives a probability for the tag sequence y:

$$p(\boldsymbol{y}|\boldsymbol{X} = \frac{e^{s(\boldsymbol{X},\boldsymbol{y})}}{\sum_{\widetilde{y} \in \boldsymbol{Y_X}} e^{s(\boldsymbol{X},\widetilde{\boldsymbol{y}})}} \tag{9}$$

Training the CRF is done by maximizing the log-probability of the the gold-label-sequence:

$$log(p(\boldsymbol{y}|\boldsymbol{X})) = s(\boldsymbol{X}, \boldsymbol{y} - log\left(\sum_{\widetilde{y} \in \boldsymbol{Y_X}} e^{s(\boldsymbol{X},\widetilde{\boldsymbol{y}})}\right) \tag{10}$$

$$= s(\boldsymbol{X}, \boldsymbol{y}) - \underset{\widetilde{y} \in \boldsymbol{Y_X}}{logadd}\ s(\boldsymbol{X}, \widetilde{\boldsymbol{y}}) \tag{11}$$

$\boldsymbol{Y_X}$ represents the set of all possible tag sequences for sentence X. In the decoding phase, the predicted sequence is obtained by taking the sequence with the highest score:

$$\boldsymbol{y}^* = \underset{\widetilde{y} \in \boldsymbol{Y_X}}{argmax}\ s(\boldsymbol{X}, \widetilde{\boldsymbol{y}}) \tag{12}$$

3 Experiments

Both models were implemented in python3 with the pytroch [Paszke et al. 2017] package. For the CRF layer a special package was used [Ikeda 2018].

The hyperparameters used for training are largely the same as they were used in [Lample et al. 2016]. The following parameters which are used for both the LSTM-model and the LSTM-CRF-model have an equal value as used in the paper: The hidden dimension of the LSTM is set to

	Articles	Sentences	Tokens	B-LOC	B-MISC	B-ORG	B-PER
Training set	946	14987	203,621	7140	3438	6321	6600
Validation set	216	3466	51,362	1837	922	1341	1842
Test set	231	3684	46,435	1668	702	1661	1617

Table 1: Overview of the english CoNLL-2003 dataset. Table inspired by [Sang and De Meulder 2003, table 1 and 2]

100; the number of layers used by the LSTM is 1; the LSTM is set to be bi-directional; the dropout probability used as input to the LSTM is set to 50 percent; the learning rate is set to 0.01.

The following hyperparameters are set to a different value than in [Lample et al. 2016]: Both models are trained with a batch size of 256 (as opposed to a batch size of 1); as optimizer ADAM [Kingma and Ba 2014] is used (as opposed to Stochastic Gradient Descent (SGD)); a L2 regularisation [Ng 2004] is set to 0.0001. It is not mentioned in [Lample et al. 2016] for how many epochs their networks are trained. In this work the number of training epochs is set to 40 because the networks have converged at this point.

3.1 Embeddings

The input to both the LSTM-model and the LSTM-CRF-model are word embeddings trained with the glove algorithm [Pennington, Socher, and Manning 2014]. In [Lample et al. 2016] 100-dimensional word vectors were used. Here, 50- dimensional word vectors are used, trained on the combined Wikipedia 2014 + Gigaword 5th Edition corpora (6 billion tokens, 400,000 tokens in the vocabulary). The 50-dimensional word vectors can be downloaded from [Tatman 2017].

3.2 Dataset

Both models, the LSTM-model and the LSTM-CRF-Model, are trained and evaluated on the english CoNLL-2003 dataset [Sang and De Meulder 2003] in the version provided by [davidsbatista 2018]. No pre-processing was performed on the dataset. Table 1 gives and overview of the number of sentences, tokens and BIO tags in the dataset.

4 Results

Both networks were trained with the hyperparameters described in section three. For the test dataset the LSTM-Model obtained an accuracy of 0.951, a weighted F1 score of 0.947 and a macro F1 score of 0.802; the LSTM-CRF-Model obtained an accuracy of 0.958, a weighted F1 score of 0.956 and a macro F1 score of 0.824. Therefore the LSTM-CRF had a 0.7 % better accuracy, a 0.9% better weighted F1 score and a 2.2% better macro F1 score. In the training dataset the LSTM-Model has a macro F1 score of 0.935 and the LSTM-CRF-Model has a macro F1 score of 0.908 which means that the LSTM-model These results are illustrated in figure 3.

In figure 4 there are confusion matrices of both models on the test set. It can be observed how well the models predict a specific tag, and how often a tag is mispredicted as an other tag. The confusion matrices are build in such a way that one row sums up to one. In other words, by looking at the entry in row i column j one can observe what percentage of the true BIO tag in row i are mispredicted

	LSTM			LSTM-CRF			
	precision	recall	F1	precision	recall	F1	support
O	0.96	0.99	0.98	0.97	0.99	0.98	42757
B-LOC	0.93	0.83	0.87	0.93	0.86	0.89	1837
B-PER	0.95	0.74	0.83	0.81	0.88	0.84	1842
B-ORG	0.83	0.68	0.74	0.87	0.75	0.80	1340
B-MISC	0.87	0.72	0.79	0.89	0.80	0.84	922
I-PER	0.94	0.82	0.87	0.88	0.92	0.90	1307
I-ORG	0.83	0.51	0.63	0.85	0.55	0.67	750
I-LOC	0.86	0.71	0.78	0.78	0.79	0.78	257
I-MISC	0.91	0.60	0.72	0.88	0.59	0.71	346
accuracy			0.95			0.96	51358
macro avg	0.90	0.73	0.80	0.87	0.79	0.82	51358
weighted avg	0.95	0.95	0.95	0.96	0.96	0.96	51358

Table 2: Per BIO-tag evaluation of both the LSTM model and the LSTM-CRF model.

as the BIO tag in column j. It can be observed that for every BIO tag except I-MISC the LSTM-CRF-model has a higher performance than the LSTM-model. The total number of BIO tags in the test dataset is 8601 and there are 42757 O-tags. The LSTM-model predicted in total 6959 BIO tags and 44399 O-tags. The LSTM-CRF-model predicted in total 8022 BIO tags and 43337 O-tags. This means that the LSTM-model predicted BIO tags only $\frac{6959}{8601} = 0.81$ times as often as they appear in the data and it predicted O-tags $\frac{44399}{42757} = 1.04$ times as often as they appear in the data. The LSTM-CRF-model predicted BIO tags $\frac{8022}{8601} = 0.93$ times as often as they appear in the data and it predicted O-tags $\frac{43337}{42757} = 1.01$ times as often as they apear in the data.

While figure 4 gives a good quick overview, more precise information on the models' performance can be taken by looking at a per-tag evaluation which is given in table 2.

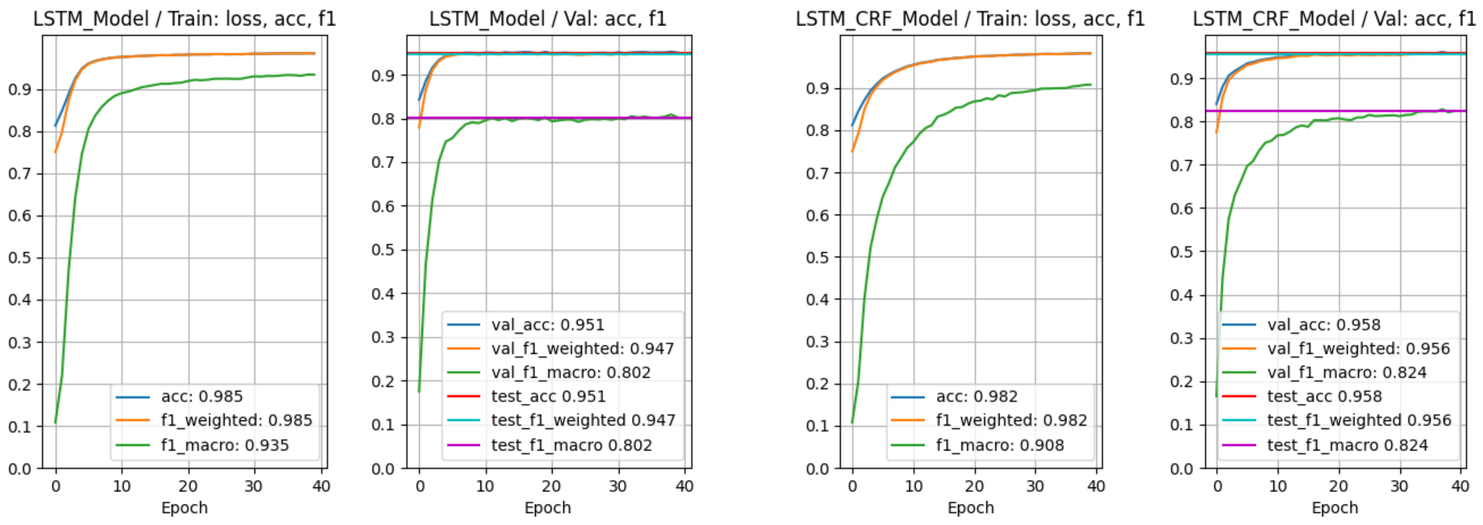


Figure 4: Accuracy and F1 score over epochs of the LSTM model and LST-CRF model for the training and validation dataset. Performance on the test dataset indicated with horizontal lines.

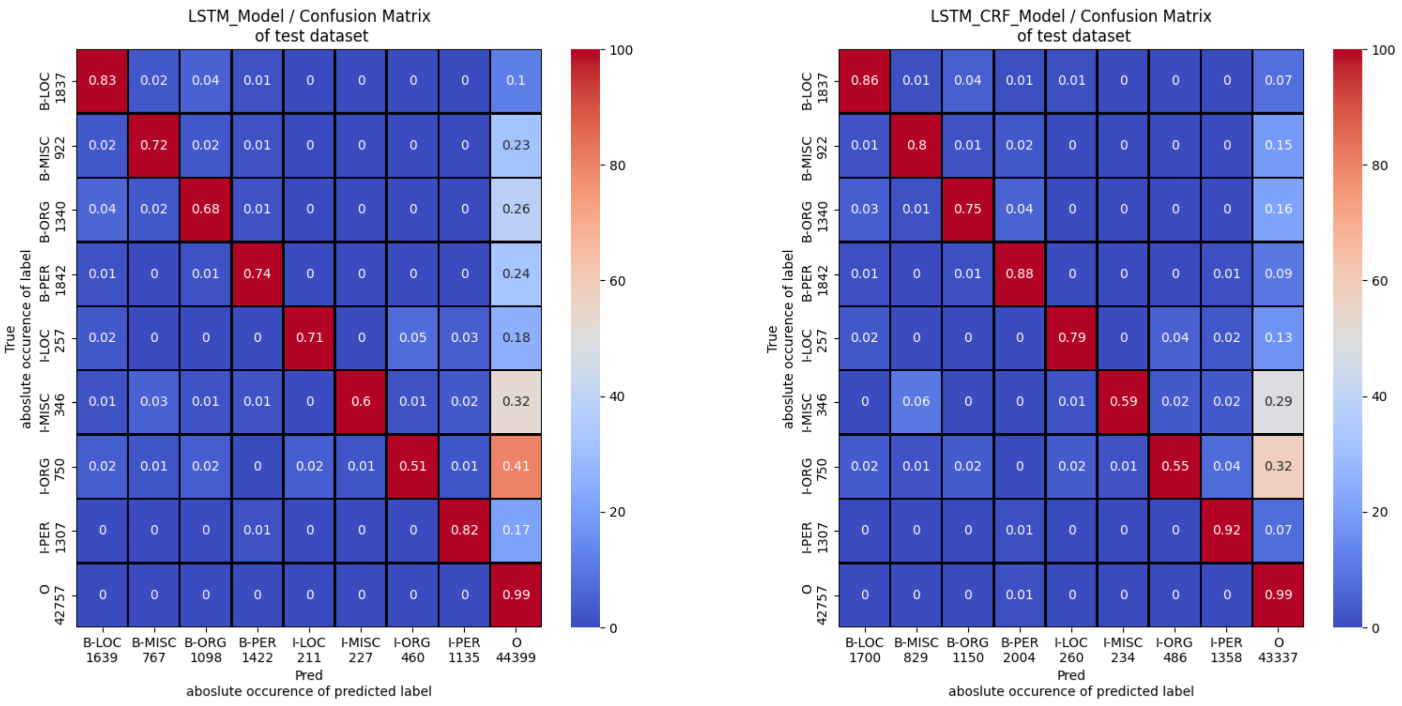


Figure 5: Confusion matrices for the LSTM and the LSTM-CRF model on the test dataset.

5 Conclusion

In this paper two models for NER were tested against each other in order to judge the efficiency gain of adding a CRF layer on top of a LSTM network's output layer. It was found that overall the CRF layer yields a performance increase. It was found that the LSTM model predicted BIO-tags only 0.81 times as often and the LSTM-CRF model predicted BIO-tags BIO-tags BIO-tags 0.93 times as often as they appear in the data. Therefore, it can be said that the LSTM-CRF model's predictions have a higher tendency to reflect the true distribution of tags vs. no tags. Judging by the fact that the LSTM-model had a higher training macro f1 score but a lower test macro f1 score one can say that the LSTM-model overfits the data while the LSTM-CRF-model generalizes better.

6 Discussion

It was shown that adding a CRF layer in overall benefits all metrics. One analysis which was not made in this work is to analyse how often both models predict bi-gram BIO-tags that do not obey the implicit grammer of the BIO tagging scheme, i.e. B-PER, I-LOC. Such an analysis could be made in an other work.

References

davidsbatista (2018). *CoNLL-2003 dataset*. URL: https://github.com/davidsbatista/NER-datasets/tree/master/CONLL2003.

Elman, Jeffrey L (1990). “Finding structure in time”. In: *Cognitive science* 14.2, pp. 179–211.

Goldberg, Yoav (2017). *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Ikeda, Ryuya (2018). *TorchCRF*. URL: https://github.com/s14t284/TorchCRF.

Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.

Lample, Guillaume et al. (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*. San Diego, California: Association for Computational Linguistics, pp. 260–270. DOI: 10.18653/v1/N16-1030. URL: https://www.aclweb.org/anthology/N16-1030.

Ling, Wang et al. (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*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1520–1530. DOI: 10.18653/v1/D15-1176. URL: https://www.aclweb.org/anthology/D15-1176.

Ng, Andrew Y. (2004). “Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance”. In: *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML ’04. Banff, Alberta, Canada: Association for Computing Machinery, p. 78. ISBN: 1581138385. DOI: 10.1145/1015330.1015435. URL: https://doi.org/10.1145/1015330.1015435.

Paszke, Adam et al. (2017). “Automatic differentiation in PyTorch”. In:

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Sang, EF Tjong Kim and F De Meulder (2003). “Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition”. In: *Proceedings of CoNLL-2003, Edmonton, Canada*. Morgan Kaufman Publishers, pp. 142–145.

Tatman, Rachael (2017). *GloVe: Global Vectors for Word Representation*. URL: https://www.kaggle.com/rtatman/glove-global-vectors-for-word-representation?select=glove.6B.50d.txt.