INDEPENDENT STUDY

NAMED ENTITY RECOGNITION WITH NEURAL NETWORKS

November 30, 2017

Student ID: 6886047
University of Hamburg
Department of Language Technology

Examiner: Prof. Dr. Chris Biemann

Contents

1 Named Entity Recognition

Named Entity Recognition (NER) describes the task of finding or recognizing named entities. Since 'entity' is a very broad term, meaning something that exists, it is concretized for this purpose. Named entity refers to either Person, Location, Organization or Misc-Entity in this context. Any other word is referred to as being no entity. One could argue, that these four types of entities do not cover the whole semantic space of possible entities, but since most of the available data is labeled in this fashion, it is used in various studies, including this one. Finding entities in text or speech is an essential preprocessing step in oder to extract further information. Almost all information in text or speech can be decomposed into entity-to-entity-relations or subject-object-relations, where subject and object are normally some form of entity. So in order to find relations in language, a distinction between entities and no-entities or other semantic word forms is necessary.

Next to this, tagging words with their respective entity-tag will increase the general richness of information provided by that word. This means, that further natural language algorithms can use these words combined with their NER tags in order to increase their performance. For example, if one wanted to determine the POS tag (noun, verb, ...) for these words now, the NER tags would provide critical information. This can also be done the other way round, which is actually more common.

NER can also be used to acquire general information about text. E.g. if it is mostly about Locations or Persons or what the composition between the different entities is.

The goal of this study is to replicate the state-of-the-art performance in NER without any handcrafted rules, but only by using pre-labled training data. In addition, approaches to further increase the performance are described and evaluated.

Similar studies that reach state-of-the art performance with neural architectures can be looked up in (Ma & Hovy, 2016) or in (Lample, Ballesteros, Subramanian, Kawakami, & Dyer, 2016).

2 Data

Two labeled data collections were used for training the networks and testing them. The first one is the CoNLL-2003 (Tjong Kim Sang & De Meulder, 2003)[1] dataset, which is probably the most widely used dataset for NER. Text is split word wise and every word is tagged with it's POS and entity-tag. In total there are nine different token-level-tags, representing four

[1]https://www.clips.uantwerpen.be/conll2003/ner/

Table 1: Table of the different datasets split up into their respective training, development and test set. The number of words with respective tags can be seen. In the second last column the total number of words per document is listed. In the last column the percentage of words with a tag for the whole document is shown. The English dataset has a higher percentage of tagged word compared to the German ones.

Model	O	B-Per	I-PER	B-ORG	I-ORG	B-LOC	I-LOC	B-MISC	I-MISC	total	No.entities/total
CoNNLEngTrain	170524	6482	4646	6110	3915	7039	1258	3357	1236	204567	**16.64%**
CoNNLEngDev	38554	1579	1194	1595	901	1645	280	683	235	46666	**17.38%**
CoNNLEngTest	42975	1816	1333	1289	803	1802	292	904	364	51578	**16.68%**
CoNNLDeuTrain	192915	2793	1735	2147	1666	4208	789	767	464	207484	**7.02%**
CoNNLDeuDev	47682	1206	634	580	425	1038	213	203	117	52098	**8.48%**
CoNNLDeuTest	46035	1393	606	1088	860	1190	148	214	111	51645	**10.86%**
GermEvalTrain	410764	7925	4498	6101	3788	11602	1189	3450	3536	452853	**9.29%**
GermEvalDev	37693	731	441	590	373	1050	151	303	321	41653	**9.51%**
GermEvalTest	87530	1694	915	1330	703v	2376	307	778	866	96499	**9.29%**

different entities: 'O' for no-entity and two different tags for each entity (Person, Location, Organization, Misc), signaling whether the entity is at the beginning or inside an entity that possibly spans over more than one word. The categories or possible token-tags look like this: 'O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC'.

Depending on the tagging schema 'B-Entity', 'I-Entity' and 'E-Entity' are used for Begin-, Inside- or End-position of the tag. In this work, the IBO2 tagging schemaWu, Yang, Lee, and Yen(2006) was used. Here 'B-Entity' is used when ever a new entity is beginning (regardless if it is longer than one word or not). 'I-Entity' is used for the words after 'B-Entity', when an entity spans over more than one word.The CoNLL-2003 dataset was used in English and German language.

The third dataset is the GermEval-2014[2] dataset. This dataset provides more information about entities (e.g. about nested entities). This information was not taken into account. It was changed to the same data format as the CoNLL-2003 dataset. By doing so, the same evaluation script could be applied to both dataset, making them directly comparable. Also the same system could then train on training sets from both datasets. Detailed information about the sizes and the distributions of tags in the different datasets and languages can be seen in Table 1.

3 Neural Networks

Neural networks (NN) consist of a number of weight matrices, used to project a given input [x] to an output [y]. By providing training data, and a loss function, optimization algorithms like gradient decent are used to adapt the weights in order to minimize the given loss function.

[2]https://sites.google.com/site/germeval2014ner/data

If the network contains more than one layer between input and output layer - called hidden layers - then a backpropergation algorithm is used to propergate the loss through the whole network. Basic knowledge about neural networks is assumed in this paper. If this is not the case, Schmidhuber(2015) and Haykin(1994) have a more detailed explanation of the underlying mechanisms and algorithms.

3.1 Recurrent Neural Networks

In order to implicitly give time- or sequence-information to a NN, recurrent neural networks (RNN) are used. RNNs get a sequence of features as input, where each feature could stand for one timestep of some process. In this study every sentence is represented as a sequence and every word is a feature. Since RNNs suffer from problems like the vanishing/exploding gradient, LSTMs are used instead. Long-short-term-memory (LSTM) cells are designed to keep important information even if the sequences are very long and important information was recognized in one of the first samples of the sequence. This is done implicitly with the architecture of LSTM cells. Every LSTM cell is build up by a number of gates (e.g. forget gate), which allow it to store important information much longer and discard unimportant one faster. Ma and Hovy(2016) explain this in more detail if needed. LSTMs are one of the most widely used forms of RNNs and NN in general. They have proven state-of-the-art performance in many sequence based problems.

3.2 Convolutional Neural Network

Convolutional neural networks (CNN) are based on NN, but realize some new ideas, especially in oder to analyze image-data. Where a regular feed forward neural network connects every node from one layer to every node of its preceding layer, a CNN only connects a certain number of nodes to one of its preceding nodes. One can understand this like applying a filter or a convolution to an image. By doing so, one saves a lot of computational time on the one hand, on the other hand local context becomes more important in comparison to a feed forward network, where always the context of the whole input is considered. CNNs have proven themselves in various image classification tasks and, like LSTMs for sequences, have become state of the art for many image/pixel based problems. Since embedding every word of a sentence into a vector creates a n*m matrix, where n is the number of words and m is the embedding size, CNNs can also be applied to embedded text data.

3.3 Conditional Random Fields

An additional Conditional Random Field (CRF) instead of the last softmax layer of one of the previously described neural networks allows to take the context of the output tags into account as well. Where the LSTM or CNN are used to extract the information from the input context, the CRF layer is doing so for the context of the output tags. This means, it can make predictions about the out-coming tag on the basis of the whole output sequence. This can be helpful, because certain combinations of output tags are not possible and should never occur. The CRF layer can detect these rules and apply them accordingly. Literature shows an increase of performance on sequence tagging tasks through adding a CRF layer as last layer of the model architecture; see Section 4.3

4 Methods

4.1 Word embedding

In order to create a numeric representation of the textual input, every word/token has to be embedded into some vector representation. The easiest approach for this is to one-hot encode every word that is present. This would results in n vectors with the size n, for n being the number of different words, and no meaningful relation between two vectors. A better solution is, to embed each word based on its context. On a very high level what is done: Words are one-hot encoded and trained on their context words (surrounding words) with one hidden layer. The weights of this hidden layer are taken as word embedding for each word respectively. More detailed information about how this can be done is provided in Mikolov, Chen, Corrado, and Dean(2013). More important for this paper than the actual implementation, are the implications from this embedding. A word embedding where words are embedded on the basis of their neighboring words allows a meaningful comparison between vectors. Since similar words occur in similar contexts, similar vectors can be interpreted as a semantic similarity. Also this method allows to embed words into smaller vectors. A very often seen embedding size is 300.

For this paper the embedding was done with different pre-trained models, all having an embedding size of 300.

Most of the experiments were done with a pretrained fasttext model[3]. Fasttext is an embedding model, that uses continuous word representations. Instead of using discrete representations of every word, "each word is represented as a bag of character n-grams. A

[3]https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

vector representation is associated to each character n-gram; words being represented as the sum of these representations"(Bojanowski, Grave, Joulin, & Mikolov, 2016)

Also Google's pretrained word2vec[4] and BPEmb[5] models have been tried out, but since they performed worse than the fasttext model, were not used for further experiments. When embedding the words, all numbers were replaced with 0 and all unknown words with a '-' in them were divided into two separate words. The '-DOCSTART-' symbol was handled as a unique token and all characters were converted to lower case (except in the word2vec model), since the fasttext and BPEmb model were only trained on lower case words. Other than that, the words were not preprocessed in any way.

At last, to get more information about the word in question, the word was separated into its characters and and every character was encoded into its ASCII value; see Section 4.4.

4.2 Bi-directional context

In order to take context information into account when predicting the tag of a word, the words before and after the word in question were also embedded and used as additional information. The sentence was split into two lists, where the first list contained all word embeddings of words from the beginning of the sentence until the word in question inclusive, and the second list contained all the embeddings from the last word to the word in question (inclusive). The target variable for one of these data points was a one-hot encoding over the 9 previously described categories of 'O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC'. An example of one data point can be seen in Figure 1.

4.3 Model architecture

The architecture of the Bi-LSTM neural network model was adjusted during the experiment phase, but most of the experiments were done with a similar architecture. After the embedding, two LSTM networks with 300 LSTM cells each were added to the model. Those were concatenated afterwards and their result was fed into two three connected dense layers with 300 and 100 and 9 units respectively. After the concatenation and the first dense layer a dropout of 0.6 was applied. If the CRF layer was not used, the softmax activation function was applied for the last layer, yielding a probabilistic distribution over the nine output categories. For all other layers the ReLu activation function was used.

[4]https://code.google.com/archive/p/word2vec/
[5]https://github.com/bheinzerling/bpemb

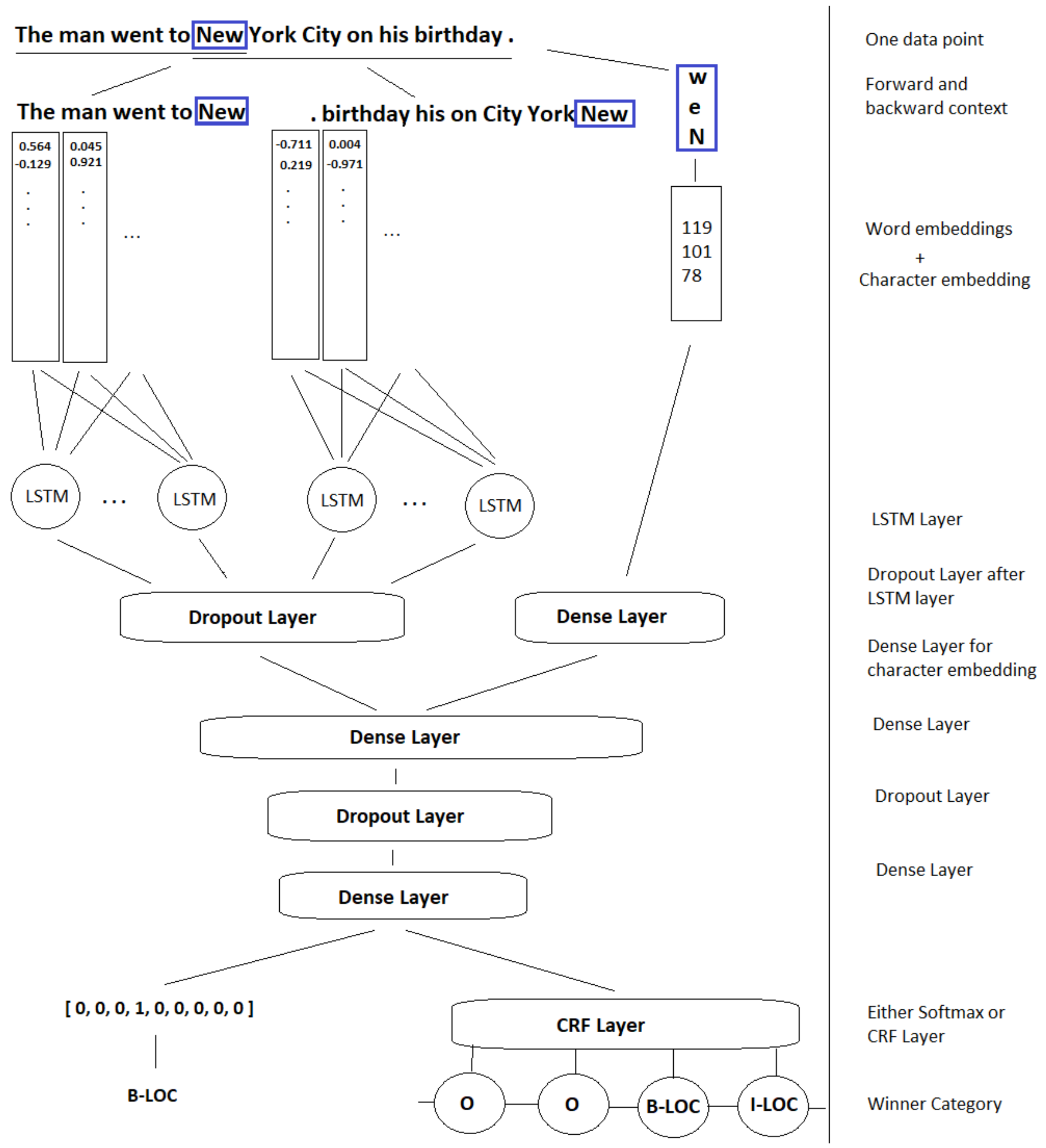

Figure 1: Architecture of the model with an example data point. The word 'New' is the word to be tagged in this example. This shows the full architecture of the model. On the right the different steps are labeled.

If the CRF layer was used, ReLu activation function was also used for the last dense layer. The last layer was then reshaped to a sequence length of one and given into the CRF layer which returned a one-hot-encoding over the nine output categories. This architecture resulted in having around 1.7 million trainable parameters. Batchsize was set to 512 for all experiments without CRF, and 300 for all others. The code is openly available.[6]

4.4 Character embedding

In order to give more importance to the actual word in question, the word was separate into its characters and the ASCII representation (a -> 97) of every character was saved in a list. This list was reversed and used as separated input (next to the bi-directional LSTM inputs), that was given into a dense layer and then concatenated with the first dropout layer after the concatenation of the two LSTM layers; Figure 1.

4.5 CNN instead of LSTM

A second architecture was experimented with. The two LSTM layers were replaced by a number of convolutional layers, spanning over a different number of words. Each convolutional layer consists of a fixed number of convolutional filters, each spanning over a matrix of 300*n, where 300 is the embedding size of each word and n is the number of words. Depending on n, the context information increases. If n would be 1, the filter would only 'see' one word at a time and have no direct context information. If n would be the same as the sentence length (defined to 20 in this case), the filter would 'see' the whole sentence and context information could be included directly. A bigger filter increases the trainable parameters and thus training time. The idea behind a CNN is therefore to use multiple smaller filters in order to capture spacial and local information (mostly in images). This can also be applied to text input, since relevant information is often (depending on the language) packed together. In this task for example: 'I-LOC' will always be next to 'B-LOC' or 'I-LOC'. Size and number of CNN layers are reported within the results below.

5 Related Work

As mentioned before, the goal of this study was to replicate state-of-the-art behavior for the NER task. The methods by which this was tried, are similar to other papers. It is not a

[6] https://github.com/fabudlx/NER

one to one replication study, but the main elements of the neural network architecture have been used in other studies before and proven to show a very good performance. The two papers that were used most importantly as reference, and are similar to the approach in this article, are Ma and Hovy(2016) and Lample et al.(2016). Lample et al. also use a Bi-LSTM with CRF layer and character embedding for their best F1-Score performance: 90.94 on English, 78.76 in German CoNLL-2003 dataset. By doing so, they increased the best F1-Score for the German CoNNL-2003 by 2.54%. They outperformed Gillick, Brunk, Vinyals, and Subramanya(2015), who used character based word representations/embeddings and used a sequence to sequence learning approach. Ma and Hovy use a combination of BI-RNN, CNN and CRF to score a F1-Score of 91.21% on the English CoNNL-2003 dataset. Using a CNN layer instead of a LSTM layer like mentioned in 4.5 was done by Collobert et al.(2011) and also showed comparable but worse results than the Bi-LSTM architectures. Huang, Xu, and Yu(2015) use a similar LSTM-CRF architecture, but they use handcrafted spelling features to increase the performance of their model. Table 2 and Table 3 show the best performance for CoNNL-2003 dataset of this paper compared to other state-of-the art approaches.

Related work with the GermEval-2014 dataset can be found in Benikova, Biemann, Kisselew, and Padó(2014). Nam(2014) and Nouvel and Antoine(2014) use forms of supervised or semi-supervised training to solve the task, but only reach F1-Scores of 72.30% or 62.85% respectively. Hänig, Bordag, and Thomas(2014) score an F1-Score of 77.14%, and have the highest found performance on this dataset. But they use external resources, like looking up unknown words in the Wikipedia API, for example, which increases their performance. These performances refer to GermEval-2014 metric 2, which are in line with the metric used for CoNNL-2003 results, and are also used for this paper. Metric 2 refers to using the same tag schema as described in Section 2. Table 4 shows a comparison on different studies on this dataset.

6 Results and Performance

The performance of the NER task is measured and shown with the average F1-Score between all the categories. Since the most words are no-entities, accuracy can lead to misinterpretation of the results. Tagging everything as no-entities yields a accuracy between 80-90% depending on the percentage of entities in all words. F1-Score on the other hand gives a more accurate picture of the actual performance in a categorization task. F1-Score is the harmonic mean between precision and recall. All F1-scores shown are calculated by the evaluation script provided for the CoNNL-2003 task. For a multi-category problem the average F1-Score is

Table 2: Shows different performances on the CoNLL-2003 **English** dataset of models from various studies

Model	F1
(Lin & Wu, 2009)	83.78
(Collobert et al., 2011)	89,59
(Luo, Huang, Lin, & Nie, 2015) + gaz	89.90
(Passos, Kumar, & McCallum, 2014)	90.05
(Huang et al., 2015)	90.10
This paper	**90.14**
(Passos et al., 2014)	90.90
(Lin & Wu, 2009)	90.90
(Lample et al., 2016)	90.94
(Luo et al., 2015) + gaz + linking	91.20
(Ma & Hovy, 2016)	91.21

Table 3: Shows different performances on the CoNLL-2003 **German** dataset of models from various studies

Model	F1
(Gillick et al., 2015)	72.08
(Ando & Zhang, 2005)	75.27
(Qi, Collobert, Kuksa, Kavukcuoglu, & Weston, 2009)	75.72
(Gillick et al., 2015)	76.22
(Lample et al., 2016)	78.76
This paper	**79.89**

Table 4: Shows different performances on the **GermEval-2014** dataset with Metric 2 (subtypes base, deriv and part collapsed)

Model	F1 (Metric 2)
(Nouvel & Antoine, 2014)	62.85
(Nam, 2014)	72.30
This paper	**77.02**
(Hänig et al., 2014)	77.14

Table 5: This Table shows the results from the **CoNNL-2003** dev and test set for different architectures and training sets.

Model	CoNNL Eng dev	CoNNL Eng test	CoNNL Deu dev	CoNNL Deu test
Bi-LSTM	83.9	88.72	**79.28**	79.49
CRF	84.07	88.36	76.67	78.53
CHAR	**86.07**	**90.14**	78.96	**79.89**
CRF+CHAR	86.01	89.6	78.12	78.84

calculated over all classes.

6.1 Model Architecture

Performance was tested with four different LSTM conditions, referring to four different model architectures. The basic architecture was the Bi-LSTM architecture described in Section 4.3 (LSTM). The other three architectures have either a CRF layer added (CRF), the character embedding of the word in question as an additional input (CHAR) or both (CRF+CHAR). The resulting average F1-Score for all four conditions on all three datasets (CoNNL-eng, CoNNL-ger, GermEval) can be seen in Table 5 and Table 6. The CHAR model shows best results over almost all categories.

6.2 CoNLL-2003

The results shown in Table 2 and in Table 3 are acquired after 15-17 training epochs on the whole training data and tested on the test data. Test data was only used once for the final test, which yielded the results shown in the tables. All fine-tuning of variables of the network architecture were made, based on the results from the development dataset.

6.3 GermEval-2014

The same models used for the CoNNL-2003 dataset were trained and tested on the GermEval-2014 dataset. The results can be seen in Table 6. The overall results for the test data are slightly worse than for the CoNNL-2003 German dataset.

6.4 Convolutional Neural Network

As described in Section 4.5 a CNN-CHAR model was also trained and tested on the CoNNL-2003 English dataset. F1-Scores are shown in Table 7. The CNN-CHAR model is build like the

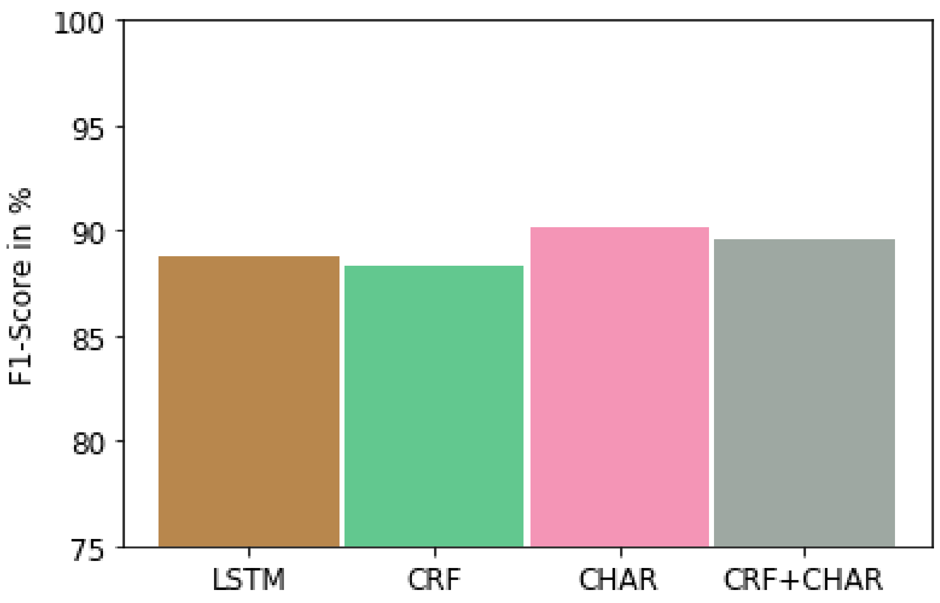

Figure 2: F1-Score for different LSTM-condition for CoNNL-2003 English test data.

Table 6: This Table shows the results from the **GermEval-2014 dev and test** set for different architectures and training sets.

Model	GermEval dev	GermEval test
Bi-LSTM	76.65	**77.02**
CRF	74.2	74.11
CHAR	**77.45**	76.94
CRF+CHAR	74.49	73.27

Table 7: This Table shows the comparison between CNN-CHAR and Bi-LSTM-CHAR for the CoNNL-2003 English dataset. Trainable parameters shows the size of the network

Model	CoNNL Eng dev dev	CoNNL Eng test	Trainable par.
CNN-CHAR	84.68	88.96	881,809
Bi-LSTM-CHAR	86.07	90.14	1,723,909

BI-LST-CHAR model, but instead of the two LSTM layers, four different CNN layers (two for forward and two for backward) with window size of 2 and 5 and 50 filters each are used. The resulting model is much smaller than the LSTM model but also performs worse than it.

7 Discussion

7.1 Character Embedding

The results show, that the best performance in the English conditions, regarding F1-Score, were observed when using a char representation of the word in question next to the Bi-directional embedding of the whole sentence with a pretrained fasttext word embedding model. Other models like the plain Bi-LSTM also perform in a similar range of F1-Score, but the adding of the character embedding always increases the performance.

For the two German conditions (CoNNL-2003 Ger and GermEval-2014) the character embedding shows an increase in performance for the test dataset of CoNNL-2003 ger and for the development set of GermEval-14. For the respective development and test set, the plain Bi-LSTM scores the highest F1-Score. Combined, the overall performance over both German conditions over development and training sets was slightly higher for the condition with character representations.

This can be explained by the increased information gain by this direct character embedding. Especially since the chosen fasttext word embedding converts everything to small letters before reading it. This completely obscures the difference between capitalized and non capitalized words. For example the word ’us’ could not be differentiated from the word ’US’. This effect is much weaker for the German condition. In German, all nouns are capitalized, where in English only proper names or proper nouns are capitalized. We can assume that this is the reason, why precise character information was more useful for the English model than for the German model.

7.2 Conditional Random Field

The CRF conditions showed inferior performance than the softmax layer, in contrary to results reported in the literature. Lample et al.(2016) could observe a rise in F1-Score of 1.79 percentage points and many other papers also experienced an addition of 1-2 percentage points in F1-Score when adding an CRF layer as last layer. A good explanation for this could not be found. Maybe more hyper-parameter tuning regarding batchsize, number of epochs and learning rate/optimizer could have given the expected rise in performance.

Hyper-parameter tuning was not done in a systematic fashion, since time and computational power did not allow for it. While a small optimization potential is to expect, further optimization was not deemed necessary in light of already good performance.

7.3 English vs. German

The clear difference in performance between English and non-English conditions was observed the same as reported in literature.

Comparing this paper to the very similar study of Lample et al.(2016) that used a similar architecture and the CoNNL-2003 dataset in German and English, one interesting difference could be observed. Lample et al. have a better F1-Score for the English condition (+0.80%), but a worse F1-Score for the German condition (-1.13%). Since both studies used the same architecture for both conditions, one possible explanation for this is the different word embedding model (fasttext) used in this paper. Since word embedding models are often trained on Wikipedia and large news corpora, the training data for English is normally much larger than for any other language. The size and diversity of a training corpus, when creating a word2vec model, using either hierarchical softmax or negative sampling is responsible for the quality of the resulting model. The explanation for the better performance of the German model compared to the slightly worse performance of the English model, in relation to other studies, is, that the German word embedding is superior to the German embedding used in previous studies. The English embedding models used in previous studies were already performing very well and the fasttext embedding used in this paper could not increase the richness of representation of words. In German on the other hand, models are generally not performing as well as English models (due to training data). A second explanation could be, that previous studies spend more time on increasing their score especially for the English model, since it has more international prestige/importance. By doing more thorough hyper-parameter tuning for example, they could have increased the score for the English models further, while they were already satisfied with the results for other languages.

State-of-the-art performance could almost be replicated in regards to the final F1-Score for the CoNNL-2003 English dataset. The model from this paper is 1.07 F1 percent points short of the best performing model right now (Ma & Hovy, 2016). This is very likely due to no systematic hyper-parameter tuning in this work and further improvements in architecture design and data-preprocessing. For the GermEval-2014 dataset, the best score without handcrafted features could be scored.

8 Conclusion

This work further proves the capabilities of neural architectures to learn tagging tasks (and categorization tasks) with no external knowledge needed. It also shows that the importance and performance of neural network architectures for natural language tasks will further increase in the future. Improved architectures and new models will further increase in accuracy and general performance in word embedding, information-inference form context and categorization tasks. This allows to solve problems faster, with less or no handcrafted features, and overall in a more general way.

References

Ando, R. K., & Zhang, T. (2005). A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research, 6*(Nov), 1817–1853.

Benikova, D., Biemann, C., Kisselew, M., & Padó, S. (2014). GermEval 2014 Named Entity Recognition Shared Task: Companion Paper. In *Proceedings of the KONVENS GermEval workshop* (p. 104-112). Hildesheim, Germany.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606.*

Chiu, J., & Nichols, E. (2016). Named Entity Recognition with Bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics, 4*, 357–370. Retrieved from `https://transacl.org/ojs/index.php/tacl/article/view/792`

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research, 12*(Aug), 2493–2537.

Gillick, D., Brunk, C., Vinyals, O., & Subramanya, A. (2015). Multilingual Language Processing From Bytes. *CoRR, abs/1512.00103.* Retrieved from `http://dblp.uni-trier.de/db/journals/corr/corr1512.html#GillickBVS15`

Hänig, C., Bordag, S., & Thomas, S. (2014). Modular classifier ensemble architecture for named entity recognition on low resource systems. In *Workshop Proceedings of the 12th Edition of the KONVENS Conference* (pp. 113–116).

Haykin, S. (1994). *Neural networks: a comprehensive foundation.* Prentice Hall PTR.

Huang, Z., Xu, W., & Yu, K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging. *CoRR, abs/1508.01991.* Retrieved from `http://dblp.uni-trier.de/db/journals/corr/corr1508.html#HuangXY15`

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (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* (pp. 260–270). Association for Computational Linguistics. Retrieved from `http://www.aclweb.org/anthology/N16-1030` doi: 10.18653/v1/N16-1030

Lin, D., & Wu, X. (2009). Phrase clustering for discriminative learning. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2* (pp. 1030–1038).

Luo, G., Huang, X., Lin, C.-Y., & Nie, Z. (2015). *Joint Named Entity Recognition and*

Disambiguation. Retrieved from `https://www.microsoft.com/en-us/research/publication/joint-named-entity-recognition-disambiguation/`

Ma, X., & Hovy, E. H. (2016). End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In *ACL (1).* The Association for Computer Linguistics. Retrieved from `http://dblp.uni-trier.de/db/conf/acl/acl2016-1.html#MaH16`

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *CoRR, abs/1301.3781.* Retrieved from `http://dblp.uni-trier.de/db/journals/corr/corr1301.html#abs-1301-3781`

Nam, J. (2014). Semi-Supervised Neural Networks for Nested Named Entity Recognition. *Proceedings of the KONVENS GermEval Shared Task on Named Entity Recognition, Hildesheim, Germany.*

Nouvel, D., & Antoine, J.-Y. (2014). Adapting Data Mining for German Named Entity Recognition. In *KONVENS' 2014* (pp. 149–153).

Passos, A., Kumar, V., & McCallum, A. (2014). Lexicon Infused Phrase Embeddings for Named Entity Resolution. In *CoNLL.*

Qi, Y., Collobert, R., Kuksa, P., Kavukcuoglu, K., & Weston, J. (2009). Combining labeled and unlabeled data with word-class distribution learning. In *Proceedings of the 18th ACM conference on Information and knowledge management* (pp. 1737–1740).

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks, 61*, 85–117.

Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4* (pp. 142–147). Stroudsburg, PA, USA: Association for Computational Linguistics. Retrieved from `https://doi.org/10.3115/1119176.1119195` doi: 10.3115/1119176.1119195

Wu, Y.-C., Yang, J.-C., Lee, Y.-S., & Yen, S.-J. (2006). Efficient and robust phrase chunking using support vector machines. *Information Retrieval Technology*, 350–361.