Context Is(n't) King: Named Entity Recognition Based Solely on Surrounding Words

Second Year Project (Introduction to Natural Language Processing and Deep Learning)

Christian Hetling, Krzysztof Parocki and **Malthe Have Musaeus**
IT University of Copenhagen
{chrhe,krpa,mhmu}@itu.dk

Abstract

The introduction of entities not seen before, called emerging entities, often severely harms performance in named entity recognition tasks. As most models base a big part of their decision on the predicted word itself, words with typos are often predicted wrong. We explore the possibility of basing the NER model entirely on the predicted word's context using Masked Language Modeling. We train and evaluate our models on the WNUT-17 dataset, a microblog-based dataset comprised mostly of emerging entities. We compare three model designs, which we denote approaches, and achieve significantly lower performance compared to the baseline model, showing that basing the named entity recognition task solely on context is not informative enough for machine learning models, contrary to expectations. [1]

1 Introduction

Named entity recognition is an important task in natural language processing and information extraction. Preserving the relation of named entities, for example when summarizing a text, lends itself to more effective summarization (Gupta, 2018). The algorithms trained for this task achieve high performance on universally acclaimed datasets (Wang et al., 2021), reaching an F1 score as high as 94.6. However, they are trained on clean datasets where the same named entities repeat, such as CoNLL 2003 and CoNLL++. The entities that are common in newly-emerging texts such as newswire or social media are often new, not having been mentioned in prior datasets. This poses a serious limitation for real-world use of leading NER algorithms and significantly harms their performance (Augenstein et al., 2017), resulting in F1 scores dropping as much as from 94.6 to 50.03 (Wang et al., 2021).

In this work, we explore the viability of using only the context of a word to classify it as a particular named entity class. Our main question is: *"Can general sentence patterns carry more information about a word being a named entity or not than the word itself?"*.

In the practical and technical sense, we want to answer the following: *"Will forcing the algorithm to use only the context, not the word itself, result in usable predictions? Is it possible to achieve better or similar performance to the leading models by training in this manner?"*. As an example, it's hard, even for human experts, to answer if "kktny" is a named entity in the following tweet: "so.. kktny in 30 mins?". We propose to look at this sentence in a different manner: "so.. [MASK] in 30 mins?".

To test our models on relevant data, we train and evaluate on the WNUT-17 dataset created by ITU researcher Leon Derczynski (Derczynski et al., 2017), comprised mostly of emerging entities in a microblog setting (such as Twitter). The dataset includes six entity classes: Person (1), Location (2), Corporation (3), Product (4), Creative Work (5), and Group (6), divided into beginnings (B) and insides (I) for entities spanning multiple words. The total number of entity types in the dataset is 12, which is 2 more entity types than in the popular CoNLL 2003/CoNLL++ datasets.

2 Previous Work

Over the years several techniques have been developed. The first automatic NER algorithms relied on handcrafted rules and part-of-speech tagging. Some of the first machine learning approaches used gazetteers, as a feature for the classifier. There have been numerous attempts to increase the performance of NER algorithms for the WNUT-17 emerging entities task (NLP-progress, 2021). They often include very complex approaches, combining character embeddings, sentence embeddings, and even representations of images accompanying the text (Shahzad et al., 2021). Zalando Research used contextualized embeddings (Akbik et al., 2019) with a modified ELMo. There have also been at-

[1] Github Repository

tempts to exploit the phonetic representation of words, given the assumption that it is more important how words sound rather than how they are written (Aguilar et al., 2018). Last but not least, there have been architectures focused on optimizing a function fed into a Conditional Random Field (von Däniken and Cieliebak, 2017). While some of these approaches do include context for classifying named entities, none of these approaches focused only on context; some of them included ablation studies, examining how sentence embeddings influenced the prediction.

3 Architecture

3.1 Baseline

For the baseline predictions, we used the DistilBERT model. We've chosen it as it was designed to be fine-tuned on various tasks, like Masked Language Modeling, and it's faster and smaller than BERT. For our task of multi-class classification, it achieved an accuracy of 94.4%, and a macro f1 score of 42.6%. The f1 score is lower than what the Huggingface community reported after fine-tuning DistilBERT. However, we decided to compare it to what we could reproduce.

3.2 Core architecture

We tested 4 different network architectures on our initial approach, containing two models: one for binary prediction ("is the masked token a named entity?") and one for multi-class NER ("knowing a named entity is here, what type of named entity is it?").

For each architecture, we define a set of shared attributes like the method used to create word embeddings and the number of linear layers before the prediction layer. Specifically, we test four models: a model with neural word embeddings and linear layers, a model with BERT word embeddings and linear layers, a model with neural word embeddings and LSTM layers, and a model with BERT word embeddings followed by LSTM layers.

The first architecture, "Uncontextualized", embeds each token into a vector with a dimension of 128. Each word embedding is passed through a linear layer that also outputs one vector per token, of dimension 128. At this point, the output of the linear layer is a tensor of dimension *(batch_size, seq_length, 128)*, but to make a prediction we want a tensor of size *(batch_size, num_entities)*. To achieve this we max-pool together each word embedding into one "sentence" vector of dimension 128 which we pass to another linear layer that, depending on the model, outputs either a binary output or a probability distribution over all entity classes.

Architecture	**Binary NER**	**Multi-class NER (without O tags)**
Uncontextualized	52	2
Contextualized	56	11
Uncontextualized + LSTM	52	5
Contextualized + LSTM	54	16

Table 1: Core Architecture MacroF1 (%) Comparison, rounded to whole numbers

The second architecture is similar to the first one, with the only difference being the word embeddings. We pass contextualized word embeddings to this model from a pre-trained DistilBERT model, prepared beforehand. We create these word embeddings by inputting each sentence to DistilBERT and summing over the last four layers, leaving us with a contextualized embedding of size 768 for each word embedding.

In the third architecture we introduce one LSTM layer after the linear layer, but before max-pooling word embeddings into one "sentence" vector. We do not pass contextualized word embeddings to this model but rather include a trainable word embedding layer in the architecture.

And finally, the fourth architecture that we test is similar to the third architecture, with the difference being that we pass it the DistilBERT contextualized embeddings.

The key metrics for each architecture, Macro F1 scores, are reported in table 1. We chose to use Macro F1 instead of Span F1 because of the ease of use coming with its native implementation in PyTorch. To evaluate our models, we were more interested in relative value compared to the baseline rather than the objective value. Therefore, it didn't matter that much that Macro F1 puts more weight to longer named entities - the baseline did that as well. At the same time, it was not a completely faulty metric, like accuracy, and therefore was fit for comparison.

We found that the best-performing architecture was the contextualized embeddings with LSTM

layers (4). It achieved a macro F1 score of 16% for multi-class, but the performance increase was lower than between the uncontextualized and the contextualized model with linear layers (2). This hints that the LSTM layer doesn't add as much to the model as the contextualized embeddings do. It even harms performance a little for the Binary NER classification, but we decided to base our decision on multi-class results as the task is much harder.

The results overall are quite low. This is the first hint that evaluating named entities based solely on surrounding words is not informative and severely harms performance instead of improving it.

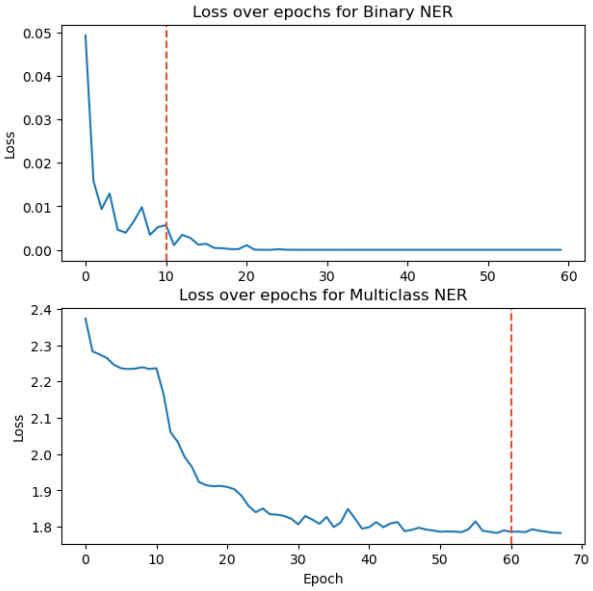


Figure 1: Loss decline over epochs, with final epoch marked as red

Hyperparameter tuning was done mainly on the fourth architecture by using an additional validation set. The Macro F1 score on the validation set reached as much as 22%, but it naturally decreased for the test set. The final hyperparameters were as follows: batch size: 32, epochs: 10 for binary, 60 for multi-class, and learning rate: 0.001 (bigger values result in the model being stuck on a local minimum, a lower value makes the training too slow), max sentence length = 32, random seed = 17. A figure of the loss values during training can be seen in figure 1

4 Methodology

After having found the best-performing architecture, we created three model designs, that we denote approaches, with the fundamental architecture and hyperparameters we just determined in the previous section. The general number of epochs was 10, training for 60 was only possible for the multi-class model of the Approach 1 model due to the less training data in the multi-class dataset.

The first approach is to train two separate neural networks. The first model takes as input a sequence where one token is masked and outputs the probability of the masked token being a named entity. The second model then takes as input the sequences that the first model predicted as containing named entities and predicts the entity for the masked token. One benefit of splitting the classification up into two models is that the second model will not be introduced to the inherent imbalance of classes because of the absence of the *nor named entity* class. The presumption for constructing a model this way is that each of the models will specialize in one task, and they will jointly achieve better performance. A visualization of the two models in this approach can be seen in figure 2.

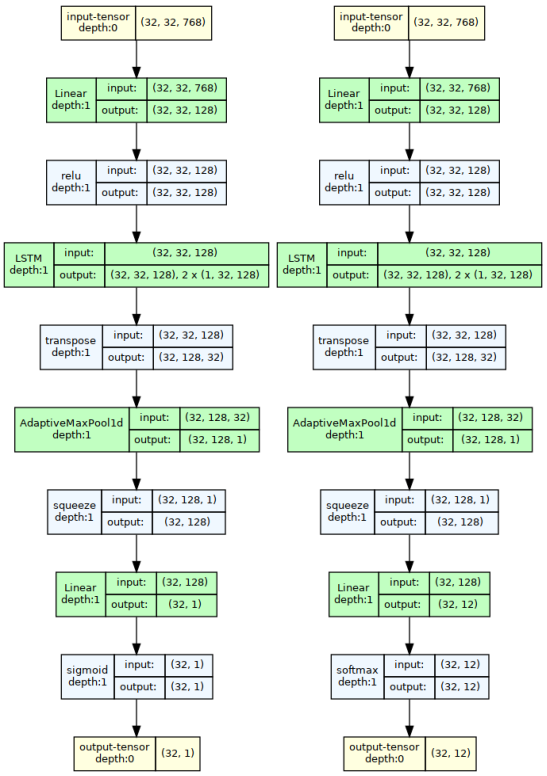


Figure 2: Neural architectures of the first approach. Binary (left), Multi-class (right).

The second approach takes as input a sequence and outputs a probability distribution over all entity classes plus an extra class being *not named entity*. Of the three approaches, this approach is arguably the most intuitive as it directly models the relation-

ship between a masked sequence and its predicted entity. A visualization of the second approach can be seen in appendix 6.

The third approach we introduce, which architecture can be seen in figure 3 is one combined model with the same sequence input, but two distinct outputs. We train the model to both predict whether a mask is a named entity and, if yes, which class it is. We achieve this by adding a gating mechanism between the output of the mask detection task and the output of the mask classification task. The gating mechanism consists of a simple linear layer with one hidden unit and a ReLU activation function that we multiply with the predicted probability distribution of entities. The idea is to provide a way for the model to learn to only run the entity classification part if the entity detection part is over a certain threshold. So, if the model learns to output 0 from this layer each time a word is not a named entity, the vector for multi-class will be zero-ed. Our motivation behind combining both entity detection and classification in one model is to let the model share weights between the tasks to prevent overfitting.

5 Analysis

	MacroF1 (binary) %	**MacroF1 (multi-class) %**
Baseline	N/A	42.6
Approach 1	54.5	16.0
Approach 2	N/A	7.0
Approach 3	54.4	8.0

Table 2: MacroF1 measures (main metric) for the baseline along with the three approaches

	Accuracy (binary)%	**Accuracy (multi-class) %**
Baseline	N/A	94.4
Approach 1	92.5	17.4
Approach 2	N/A	92.0
Approach 3	92.5	92.8

Table 3: Accuracy measures (secondary metric, showing dataset imbalance) for the baseline along with the three approaches

Using the pre-trained DistilBERT model we created a baseline to compare approaches against. The Huggingface model, while not state-of-the-art, provides us with a reproducible baseline to test against.

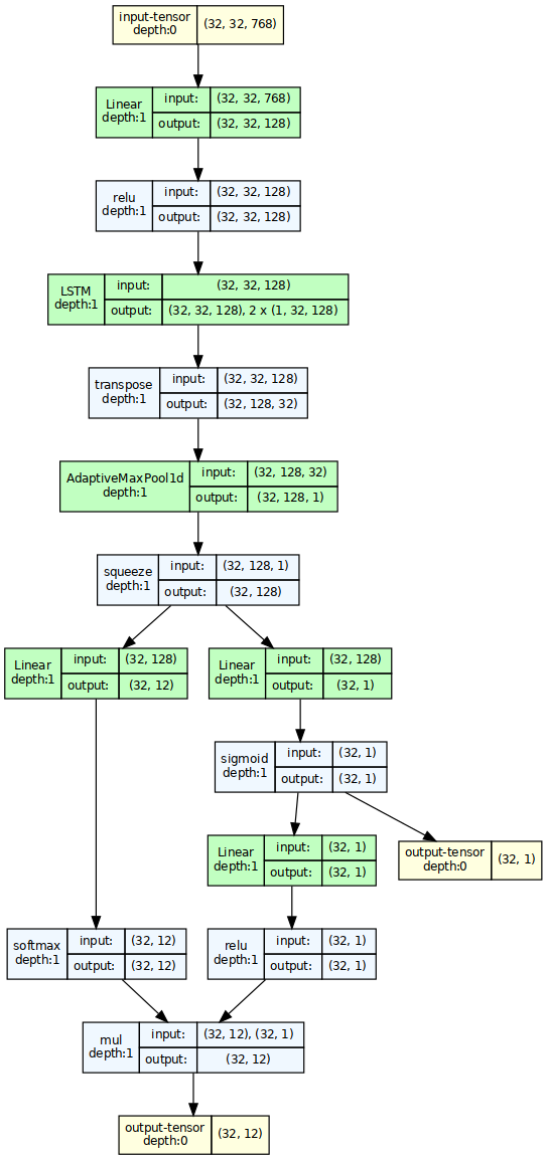


Figure 3: Neural architecture of the third approach

Note that the multi-class F1 score for approach 1, as seen in table 2, might be a bit misleading as it is only the F1 score for the second model in approach 1. Since the second model is only being fed the sequences that the first model predicted as containing a named entity, the F1 score for the second model will not take false negatives into account. The false negatives disregarded by the first model will never be passed to the second model and thus not taken into account when calculating the F1 score.

Interpretability is a key aspect of any machine learning model. It is important to understand how the model is making its predictions. Masking out the token that the model should predict, and thus only giving sentence context to the model, gives

us an insight into how the model is making its predictions. While it is hard to explain why a model is making a certain prediction, limiting what data the model has access to when making its prediction, lets us at least know the decision was made only with that knowledge.

6 Conclusion

By comparing the three approaches, we can see that specializing the models in their respective tasks was the best solution among our approaches. Both Approach 2 and 3 faced problems related to the dataset being unbalanced, which can be clearly seen from the accuracy being relatively high for all approaches. All models whose output contained the *not named entity* class reported higher accuracy (Approaches 2 and 3, 92.0 and 92.9 respectively).

Generally, the performance our models achieve is significantly lower compared to the baseline, with the best Macro F1 of 16.0 compared to 42.6 for multi-class classification. Context is therefore not informative enough to make machine learning models make good predictions. Our model is capable of making acceptable predictions in a binary manner, detecting whether or not a given word is a named entity (Macro F1 of 54) based on its surroundings. However, determining the nature of the entity proved complicated without access to the word itself.

References

Gustavo Aguilar, Adrian Pastor Ló pez Monroy, Fabio González, and Thamar Solorio. 2018. Modeling noisiness to recognize named entities using multitask neural networks on social media. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics.

Alan Akbik, Tanja Bergmann, and Roland Vollgraf. 2019. Pooled contextualized embeddings for named entity recognition. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 724–728, Minneapolis, Minnesota. Association for Computational Linguistics.

Isabelle Augenstein, Leon Derczynski, and Kalina Bontcheva. 2017. Generalisation in named entity recognition: A quantitative analysis. *Computer Speech Language*, 44:61–83.

Leon Derczynski, Eric Nichols, Marieke van Erp, and Nut Limsopatham. 2017. Results of the WNUT2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 140–147, Copenhagen, Denmark. Association for Computational Linguistics.

Shashank Gupta. 2018. Named entity recognition: Applications and use cases. Last accessed May 26th, 2023.

NLP-progress. 2021. Named entity recognition | nlp-progress. Last accessed May 25th, 2023.

Moemmur Shahzad, Ayesha Amin, Diego Esteves, and Axel-Cyrille Ngonga Ngomo. 2021. Inferner: an attentive model leveraging the sentence-level information for named entity recognition in microblogs. volume 34.

Pius von Däniken and Mark Cieliebak. 2017. Transfer learning and sentence level features for named entity recognition on tweets. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 166–171, Copenhagen, Denmark. Association for Computational Linguistics.

Xinyu Wang, Yong Jiang, Nguyn Bách, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021. Automated concatenation of embeddings for structured prediction. pages 2643–2660.

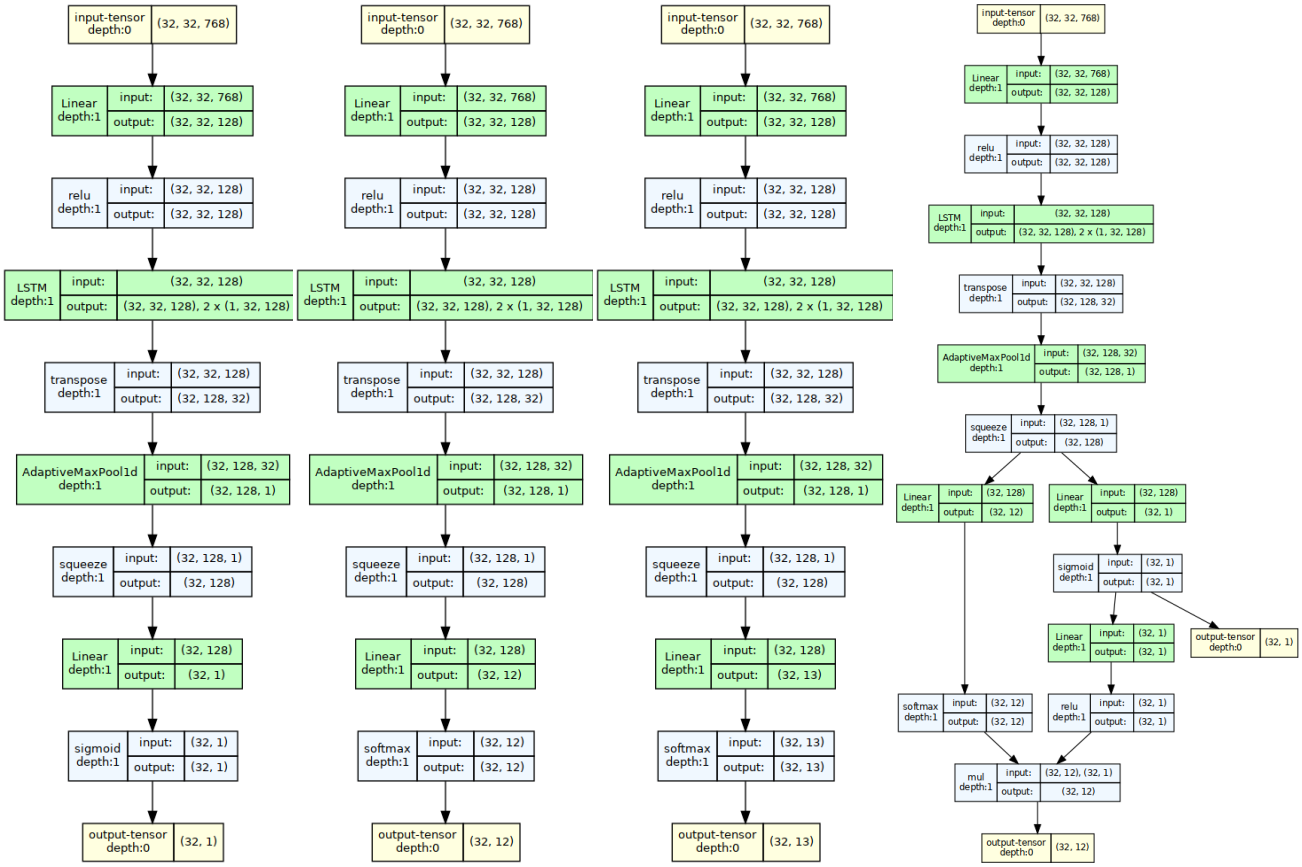
Appendix A: Group Contributions

Equal contribution. Krzysztof came up with the masked model idea in a discussion with Christian, and prepared the project in the initial stage by testing various baselines with Christian and refining the project proposal with Malthe. Christian coded data preprocessing of the WNUT-17 dataset by converting training instances to masked sequences. Malthe came up with and implemented the idea of developing our project by testing three neural architecture approaches, discussed it heavily with Krzysztof and finished the code together with Christian. Krzysztof wrote the Distilbert baseline used in the final report, and tested and evaluated different architectures the core architecture written by Malthe. The LSTM and Combined architectures were prepared together by Christian and Malthe. Krzysztof came up with the final research question and made sure we meet all requirements. Malthe prepared the architecture visualizations. The report was prepared jointly.

Appendix B: Usage of Chatbots

Chatbots were not used in the development of this project.

Appendix C: Neural Architectures



(a) Approach 1, binary (b) Approach 1, multiclass (c) Approach 2 (d) Approach 3

Figure 4: Overview of model architectures