Evaluation and re-usable implementation of DL-based approaches for Entity Recognition

Master Thesis

Universität Bonn

Supervisor: Dr. Jens Fisseler

First Examiner: Prof. Dr. Jens Lehmann

Second Examiner: Prof. Dr. Sven Behnke

Gugu Andel

Matriculation number : 3198522

July, 2021

Declaration of Authorship

I hereby declare and confirm that this thesis has been written independently by myself and that none other than the specified sources and aids were used, and that any citations have been marked.

Gugu Andel

Signed: ____

Place, Date: ____

Acknowledgements

I wish to thank everyone without whom I would not have been able to complete this project. First and foremost, I would like to express my gratitude to my supervisor, Dr. Jens Fisseler, whose knowledge and suggestions guided me through the whole process. I am extremely grateful for all the productive meetings. Special thanks to my colleagues, Sogol H., Lukas W. and Prabhu C. for offering their help and support, when I needed it.

I wish to extend my special thanks to my professors at the Computer Science department, whose lectures helped me further develop my knowledge and skills in the field of Intelligent Systems.

I can not forget to thank my family for supporting me financially and emotionally during these three long years away from home. Without them, I would not be where I am today. This thesis is dedicated to them.

Abstract

Named-entity recognition (NER) aims to identify and label instances of predefined entities in a chunk of text. Even though conceptually simple, it is a challenging task that requires some amount of context and a good understanding of what constitutes an entity. Being a precursor to other natural language applications such as question answering and text summarization, it is essential to have high-quality NER systems. For a long time, they have relied on domain-specific knowledge or resources such as gazetteers to perform well. In the past decade, deep learning (DL) techniques have been applied to NER. They do not resort to any external resources and have achieved state-of-the-art results.

This thesis investigates several aspects of DL-based approaches for NER. Recent improvements mainly come from utilizing unsupervised language model pretraining to produce representations depending on a word's contextual use. Intuitively, more informative embeddings lead to better generalization, that is, detecting mentions that do not appear in the training data for NER models. Several word representations are evaluated for German, using the two biggest available datasets CoNLL-03 and GermEval. The results show that recent contextualized representations improve the entity extraction performance on both datasets, due to being more robust against entity type ambiguities (e.g. Is "Washington" a person or a location?) or lengthy entities (e.g publication titles).

Such embeddings are useful for multilingual NER too. Two approaches for generating multilingual embeddings are pitted against each other, in order to find out which is the most useful for extracting entities in a mix of German, English and Dutch data.

Another investigated aspect is improving the performance on "low-data" domains through transfer learning, using finetuning. This is motivated by the fact that neural models tend to underperform, due to lack of sufficient data. Finetuning a pre-trained model using contextualized embeddings significantly improves the performance on a relatively small annotated German dataset from Europarl.

The final step of this work is providing a re-usable implementation of a DL-based NER model within a framework for building NLP pipelines such as DKPro Core. Challenges of integrating an external Python-based model in a Java-based framework are investigated.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI) that enables computers to understand human language in the form of text and speech. It is the engine of software that perform translations between languages, respond to questions or spoken commands and summarize large amounts of text in a short amount of time. NLP can be found around us in many forms, such as virtual assistants, chatbots, translation engines and different consumer-related services. It plays an important role even in enterprise solutions that improve employee performance and engagement, but also simplify business processes.

Some of the different tasks and applications possible with current NLP techniques are:

Speech recognition Converting voice data into text data. It is useful for applications responding to voice commands or answer spoken questions.

Part-of-Speech Tagging Determining the part of speech of a word in a sentence, based on its use, e.g. "gehe" as a verb in "Ich gehe zum Supermarkt.".

Word sense disambiguation Selecting the meaning of a word with multiple meanings, based on the context it is being used. For example, in "Du bist nicht von der Steuer befreit." Steuer means tax and in "Er drehte das Steuer nach rechts." it means steering wheel.

Named-entity Recognition Identifying proper nouns such as people names, places, brands etc. Such nouns are often referred to as entities. Examples of entities are "Deutschland" as a location or "Leslie" as a person.

Coreference resolution Finding all the expressions that refer to the same entity. For example, "Nordrhein Westfalen" and "NRW" refer to the same state in Germany.

Sentiment Analysis Extracting feelings and emotions from text.

Natural Language Generation (Text-to-Speech) Converting text data into voice data.

Recognizing and extracting entities is a fundamental task and a core process of NLP. Named-entity recognition (NER), sometimes referred to as entity chunking, extraction, or identification, is the first step in many tasks that extract information from text data. It is particularly useful for building a high-level overview in "large corpora" scenarios. References to the same entity found in multiple documents help us establish connections between documents and group them based on their subject similarity.

Use cases of NER include:

Search and recommendation engines By detecting mentions of entities in reviews, discussions and other descriptive texts, NER improves the speed and relevance of search results and recommendations.

Content classification Entities extracted by NER can be used to identify the subjects and themes of different articles, making it easier to classify their content.

Human resources NER is used to summarize applicants' CVs and categorize employee questions and complaints.

Academia NER highlights key terms and topics in academic corpora, helping students and researchers find what they are looking for faster and easier.

Healthcare NER extracts relevant information from lab reports, prescriptions etc. This reduces workload and provides better patient care standards.

The term *Named-Entity Recognition* was born in the Message Understanding Conferences (MUC), which influenced IE research in the US in the 90s. At the time, they were focusing on tasks where structured information of company activities and defense related activities was extracted from unstructured text, such as newspaper articles [19]. It became obvious that recognizing information units like names of people, locations and organizations, and numeric expressions like date, time and money was important.

The ever increasing amount of information and IE applications required the expansion of named-entity (NE) categories. The categories to be extracted depend on the target information class of IE. The wider the task becomes, the more NE categories we need [48]. In biomedicine, we might need to recognize names of proteins or genes; in news and articles, we might be more interested in names of people, organizations or locations, maybe even dates and time. NEs can be generic (person, location, organization) or domain-specific (proteins, enzymes, genes)

Early NER systems have achieved good performance, thanks to the human designed domain-specific linguistic rules and features. In recent years, deep learning, empowered by word embeddings and semantic clues, has been successfully applied in NER. With minimal feature engineering, DL-based NER systems have achieved state-of-the-art performance [48]. A considerable number of studies have improved the NER results on many languages, by proposing new DL-based architectures. Recent models rely on semantics and can generalize for multiple domains and languages [2][35].

The NER performance is affected not only by the choice of the network architecture, but also by the language itself. A lot of factors must be considered, such as morphology, similarity to other languages or amount of available data. As German is a wide-spread and comparatively well-resourced language, German NER has received its fair share of attention. Several state-of-the-art DL-based approaches have been introduced in the past few years. It is interesting to see that their performance on German texts has been traditionally lower than on other languages, such as English, Polish, Dutch etc. Studies suggest that this is due to complex linguistic structures such as compounds, separation of verb prefixes and use of uppercase letters not only for proper nouns but also regular nouns [5].

1.1 Motivation

Compared to English, there is less work on German NER. Currently, there are two big annotated datasets in German available to be used as benchmark datasets. Recent studies report scores on GermEval dataset [4], while older papers have evaluated their approaches on the German part of CoNLL-03 dataset [45].

Lample et al. [25] introduced a neural architecture based on bidirectional LSTMs and conditional random fields and trained it on four languages, including German. The model used no language-specific features or other external resources. Later, Riedl & Pado [40] showed that such a model could benefit from training on multiple domains, thus improving Lample's results. Recent improvements in many NLP tasks stem from the modern embeddings, which are extracted from pre-trained deep language models [11][2]. They incorporate contextual information and reduce the dependency of neural networks on domain- or task-specific data. In order to generalize well, NER models have to see training examples from various domains. Even though German is a relatively well-resourced language, the number of annotated datasets for NER is small and they are mainly articles from Wikipedia or daily newspapers. On scarce-data domains, transfer learning has shown to improve results for NER, but utilizing contextualized embeddings in the process might be even more useful.

1.2 Contributions

This thesis makes the following contributions:

- A comparative evaluation between a set of word embeddings. Some of them have not been evaluated before on all the available domains. New performance scores are reported. More specifically, this work quantifies the impact of GBERT and GELECTRA on CoNLL-03, character embeddings on GermEval and multilingual BERT on both datasets.
- Application of transfer learning for improving the performance on "low-data" settings. Previous work has applied transfer learning on a BiLSTM-CRF model that exploits only pre-trained word embeddings and character-level embeddings. This work includes recent contextualized embeddings on a different dataset, Europarl.
- A single NER model for German, English, Dutch using multilingual BERT to generate word representations.
- A reusable implementation of a DL-based NER model, as a custom tagger within DKPro Core, a framework that collects third-party NLP tools and makes them interoperable. This process encounters challenges and limitations, which are discussed after building up an NLP pipeline that includes one of the pre-trained NER models from the earlier comparative evaluation.

Chapter 2 describes in detail the concept of Named-entity Recognition and the deep learning techniques applied in solving the task. The last section of the chapter focuses on German NER, mainly on the available datasets and related work from recent years. Chapter 3 and Chapter 4 describe the chosen methodology for the research, experiment setups and discuss the results. Chapter 5 identifies the need for making deep learning models part of NLP pipelines and demonstrates an integration of a Python-based NER model within DKPro Core. Finally, Chapter 6 summarizes everything and suggests future work or research directions.

Chapter 2

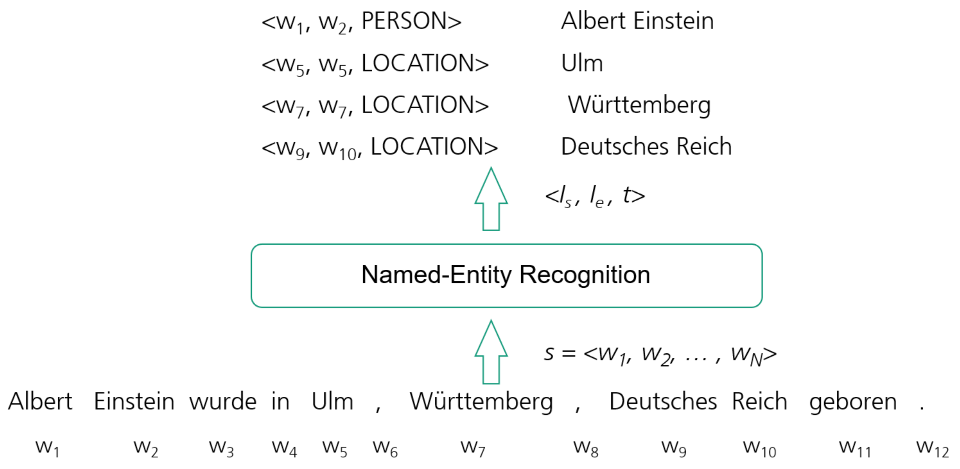
Named-entity Recognition

2.1 Definition of NER

In the field of IE, an entity is a word or a phrase that identifies an item from a set of items with similar properties. What constitutes an entity type is task-specific; people, places, and organizations are common, but gene or protein names might be relevant for some tasks.

The task of NER in NLP can be described as finding each mention of a named entity in a text and label its type. The term NE is sometimes extended to include things that aren't real-world objects, including dates, times and even numerical expressions like prices.

Formally, given a sequence of N tokens $s =< w_1, w_2, ..., w_N >$, NER outputs a list of tuples $< I_s, I_e, t >$, each of which is a named entity mentioned in s. Here, $I_s \in [1, N]$ and $I_e \in [1, N]$ are the start and the end indexes of an NE mention; t is the entity type from a predefined category set (see Figure 2.1).



<w1, w2, PERSON> Albert Einstein
<w5, w5, LOCATION> Ulm
<w7, w7, LOCATION> Württemberg
<w9, w10, LOCATION> Deutsches Reich
<Is, Ie, t>
Named-Entity Recognition
s = <w1, w2, … , wN>
Albert Einstein wurde in Ulm , Württemberg , Deutsches Reich geboren .
w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12

Figure 2.1: NER task illustration

Once all the NEs in a text have been extracted, they can be linked together in sets corresponding to real-world entities, inferring, for example, that mentions of *Nordrhein-Westfalen* and *NRW* refer to the same location. This is useful for linking text to information in structured knowledge sources like Wikipedia.

In addition to their use in extracting events and the relationship between participants, NEs are useful for a variety of language processing tasks, i.e. in sentiment analysis we might want to know a consumer's sentiment towards a particular entity. Entities are a useful first stage in semantic search. Semantic search refers to a collection of techniques, which enable search engines to understand the concepts, meaning, and intent behind the queries from users. About 71% of search queries contain at least one NE [20]. Recognizing NEs in search queries would help us to better understand user intents, hence to provide better search results. To incorporate entities in search, entity-based language models, which consider individual terms as well as term sequences that have been annotated as entities (both in documents and in queries), have been proposed by Raviv et al. [37].

Despite being conceptually simple, NER is not an easy task. The category of an NE is highly dependent on textual semantics and its surrounding context. There are two types of ambiguities that make NER difficult:

- **Ambiguity of segmentation**. Deciding what is an entity and what is not, as well as where the boundaries are, can be difficult at times.
- **Type ambiguity**. The mention *Bayern* can refer to a state in Germany, a football team, or a song from a band (see Figure 2.2).

[Bayern $_{LOC}$] ist das flächengrößte der 16 Länder in Deutschland.
[Bayern $_{ORG}$] hat in 2019-2020 insgesamt 160 Tore geschossen.
[Bayern $_{MISC}$] ist ein Lied der Band Die Toten Hosen und erschien erstmals auf dem Album Unsterblich am 6. Dezember 1999.

Figure 2.2: Example of type ambiguity in the use of name "Bayern"

2.2 Traditional approaches for Named-Entity Recognition

Before deep neural architectures were introduced, the approaches to NER were based on rules, feature engineering and other supervised or unsupervised learning algorithms. I give a short description for each method, without diving into details, as the focus is on the evaluation of the recent DL-based approaches.

2.2.1 Rule-based Approaches

A rule-based NER system uses a set of rules written by linguists which are language-dependent and help in the identification of entities in a document. For example, nouns starting with a capital letter and followed by the word "GmbH" are very likely to be organizations. Rules are also used to identify entities which are not listed in the dictionary. A dictionary, also known as a gazetteer, consists of multiple lists containing names of entities, such as locations, organizations, colors, etc. In NER, a dictionary is used to find occurrences of names in text. However, dictionaries can not store all the possible names. For example, if we want to detect unusual people names, we use rules like "Mr. XXX" to find and tag them. Rules can be designed, based on domain-specific gazetteers [14][42] and syntactic-lexical patterns [52], which assign a label to the entity if the pattern is matched. Rule-based systems work very well when lexicon is exhaustive. Due to domain-specific rules and incomplete dictionaries, high precision and low recall are often observed from such systems. Furthermore, the systems cannot be transferred to other domains. In other words, they will detect instances that they have seen before, but will fail to recognize unseen ones.

2.2.2 Unsupervised Learning Approaches

In unsupervised learning, the algorithm finds structure in its input on its own, as the input data has no labels. An unsupervised model can discover hidden patterns in data or learn features. Clustering is a typical approach of unsupervised learning, where groups of similar examples within the data are discovered. Clustering-based NER systems extract named-entities from the clustered groups based on context similarity. Collins et al. [10] and Nadeau et al. [33] proposed some of the most popular unsupervised systems that resorted to generic extraction patterns, terminologies, corpus statistics (e.g. inverse document frequency) and shallow syntactic knowledge (e.g. noun phrase chunking).

2.2.3 Supervised Learning Approaches

In supervised learning, a system receives example inputs and their desired outputs, in order to learn a general rule that maps inputs to outputs. NER is treated as a multi-class classification or a sequence labeling task. The annotated data samples are represented by carefully designed features. A feature is an abstraction over text where a word is represented by one or many Boolean or numeric values [43]. Word-level features (e.g. case, morphology and part-of-speech tag) [53][44][26], list lookup features (e.g. Wikipedia gazetteer and DBpedia gazetteer) [31][23][46], and document and corpus features (e.g. local syntax and multiple occurrences) [36][54] have been widely used in various supervised NER systems. Some of the machine learning algorithms, trained via supervised learning are:

Hidden Markov Model A generative model that assigns a joint probability to paired observation and label sequence. It uses features like capitalization, trigger words, previous tag prediction, bag of words, gazetteers, etc. to represent simple binary relations and these relations were used in conjunction with previously predicted labels. Bikel et al. [6] used HMMs to build an NER system, called IdentiFinder, to identify and classify names, time expressions, dates and numerical quantities.

Support Vector Machine A discriminative model that is trained on both positive and negative examples to learn the distinction between two classes. Yamada et al. [50] proposed the first SVM-based NER system for Japanese, which used some contextual information of the words as well as orthographic word-level features. McNamee and Mayfield [30] used language-related, orthography and punctuation features to train SVM classifiers.

Conditional Random Field One of the most prominent approaches used for NER. A linear chain CRF confers to a labeler in which tag assignment for the current word depends only on the tag of the previous word. McCallum and Li [29] exploited the ability of CRFs to take context into account along with multiple features per word of the sentence and built a CRF-based NER system.

2.3 Deep Learning-based Approaches

Deep learning is a subset of machine learning in artificial intelligence (AI) that imitates the way the human brain works to process data for detecting objects, recognizing speech, translating languages and making decisions. With the increase in the amount of information in all forms and from every region of the world, AI systems are being increasingly adapted for automated support. They process large amounts of unstructured data that would normally take decades for humans to understand and process.

Deep learning utilizes a hierarchical level of artificial neural networks, which are built like the human brain, with neuron nodes connected together like a web. While traditional machine learning models build analysis with data in a linear way, the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach.

Figure 2.3 illustrates a basic feedforward neural network. The leftmost layer is called the input layer and the rightmost layer is the output layer. The middle layers are called hidden layers because their values are not observable in the training set. Hidden layers are the place where the modeling of complex data happens. The more hidden layers a network has between the input and output layer, the deeper it is. In general, any ANN with two or more hidden layers is referred to as a deep neural network.

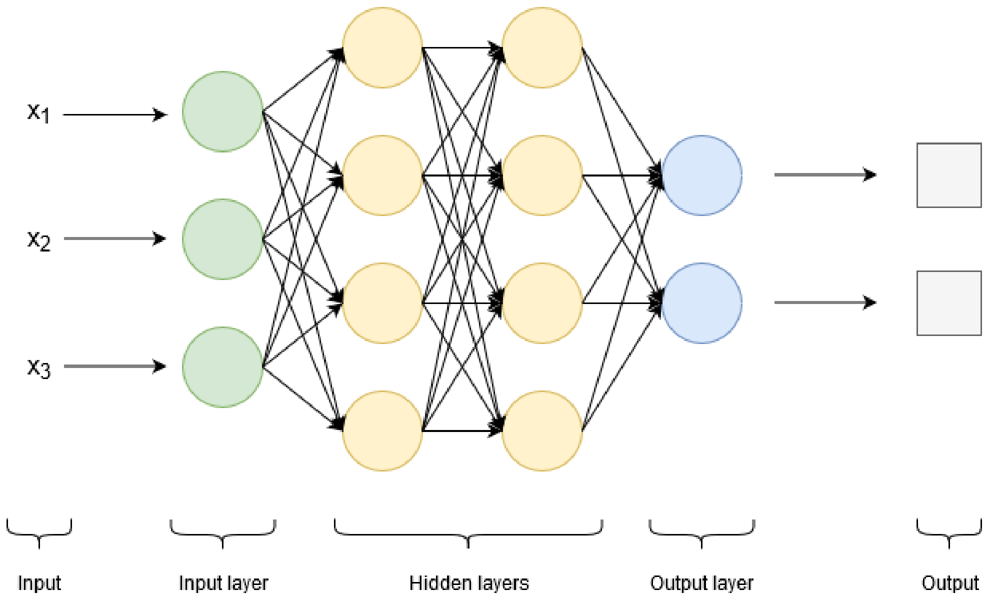


Figure 2.3: Structure of a simple feedforward neural network

DL-based architectures have successfully tackled a broad set of NLP tasks, e.g. sentence classification, question answering and sentiment analysis [11][51]. Their ability to learn complex features from data, instead of relying on engineered features, has been beneficial for NER too.

A typical DL-based NER model consists of three main components: distributed representations that map each input word to a high dimensional vector, a context encoder to capture the context dependencies between the words of a sentence and a tag decoder to predict the tags for each word (Figure 2.4). These components can be implemented in various ways, so it's necessary to explore them one by one.

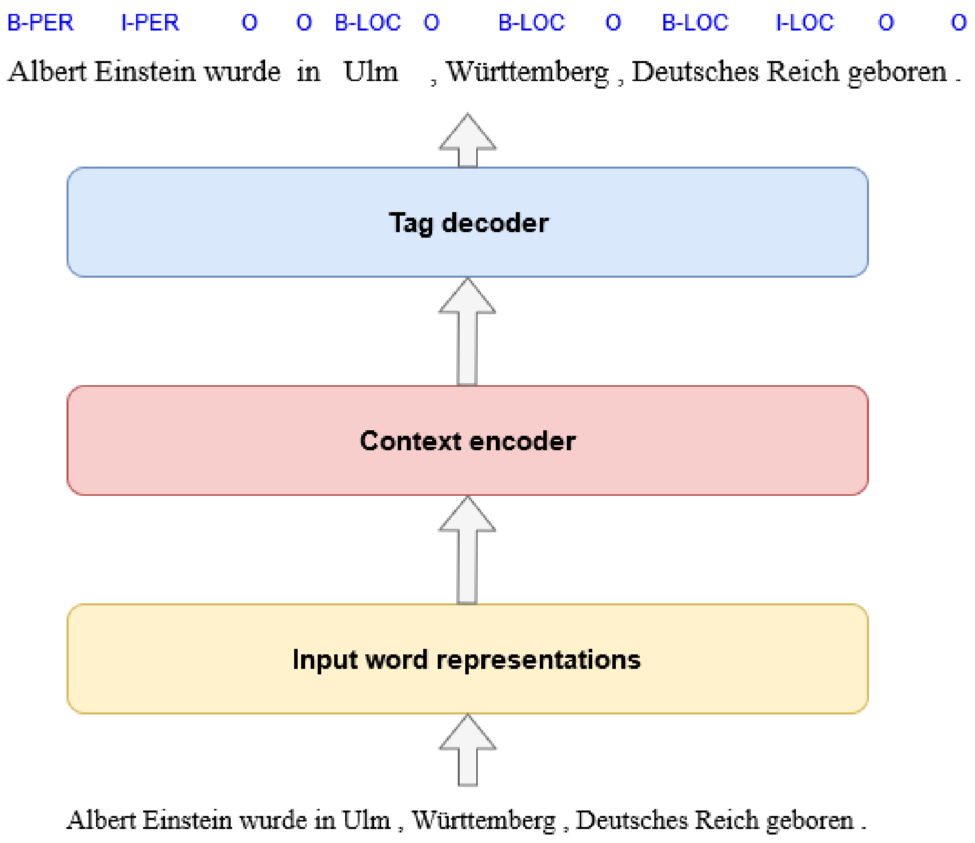

B-PER I-PER O O B-LOC O B-LOC O B-LOC I-LOC O O
Albert Einstein wurde in Ulm , Württemberg , Deutsches Reich geboren .
Tag decoder
Context encoder
Input word representations
Albert Einstein wurde in Ulm , Württemberg , Deutsches Reich geboren .

Figure 2.4: Building blocks of a typical DL-based NER model

2.3.1 Input word representations

A fair share of the success of deep learning models on many NLP problems is due to word embeddings. They are learned representations for text, such that words with similar meanings have similar representations. Word embedding methods map a fixed size vocabulary of words from a corpus into real-valued vector representations. The learning process can be either unsupervised (using document statistics, word-word co-occurrences) or joint with a neural model on a specific NLP task.

There are four types of word embeddings that are used in NER models: word-level, character-level, contextualized and hybrid embeddings.

Word-level embeddings Generated from unsupervised learning algorithms, such as continuous bag-of-words (CBOW) and continuous skip-gram models, over large collections of text [32]. Their intuition is building global word embeddings, by listing the unique words appearing in the documents and learning similar representations for words that share similar characteristics. For example, GloVe word embeddings encode semantic information by exploiting the ratios of word-word co-occurrence probabilities in a given corpus [34]. Another type of embeddings, FastText, encode morphology, by taking subword information into account [7]. These embeddings are usually pre-trained and available for free, under a permissive license.

Character-level embeddings They learn representations specific to the relevant task and domain, instead of hand-crafted prefix and suffix information about words. They are used to capture orthographic sensitivity [25]. While some compositional relationships exist, e.g. morphological processes such as adding *-lich* to a stem have relatively regular effects (end-endlich), many words with lexical similarities have different meanings, such as, the word pairs Welt-Zelt. They have been found to be useful for handling rare words and spelling errors which are usually OOV[1] for word embedding models. Sequence labeling tasks like part-of-speech tagging or entity recognition can benefit from them [27][25].

Contextual embeddings The most recent type of embeddings, which have shown to be effective for NER [2]. Unlike word-level embeddings, they are generated from methods that learn sequence-level semantics by considering all the words in a sentence or document. That means that the same word will have different representations, based on its contextual use. A more detailed description of the methods is given in the next section.

Hybrid embeddings Combinations of different embeddings and word features. In order to improve the performance of their models, some studies incorporate additional information (e.g. part-of-speech tags, gazetteers, capitalization features) into the final representation of words [28]. Although this method might lead to higher scores for DL-based models on a particular domain, it might hurt their generality, as they rely on additional word features. Another strategy is concatenating different types of embeddings (stacked embeddings). The selection of embeddings varies depending on the task; they have shown to complement one another [2].

[1]Out-of-vocabulary words, words that are not part of the training text data

2.3.2 Context Encoders

Words surrounding a word or a passage can throw light on its meaning. Given these sentences, "Ich fahre nach Washington" and "George Washington war der erste Präsident der Vereinigten Staaten von Amerika.", one can easily identify Washington as a location in the first sentence and as a person in the second, based on the surrounding words. In sequence labeling tasks such as NER, contextual information can be used to enhance the models and make better-informed tagging decisions [25]. To extract and encode context, a sequence of words needs to be processed. Typically, recurrent neural networks and transformers are used to deal with sequences and learn long-term dependencies.

Recurrent Neural Network

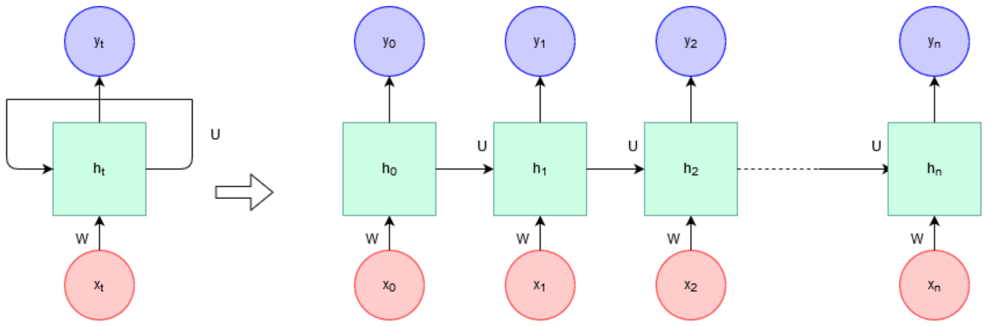


Figure 2.5: An unrolled RNN model

Unlike deep feedforward neural networks, where the output of a layer is fed as input to the next layer, recurrent neural networks (RNNs) include a feedback loop (see Figure 2.5). At time step t, they take the input vector $x_t \in \mathbb{R}^n$ and the hidden state vector $h_{t-1} \in \mathbb{R}^m$ to produce the next hidden state h_t in the following way:

$$h_t = \sigma(Wx_t + Uh_{t-1} + b)$$

where $W \in \mathbb{R}^{m\times n}, U \in \mathbb{R}^{m\times m}, b \in \mathbb{R}^n$ are parameters of an affine transformation and σ is a non-linear transformation function. The network will encode x_t to y_t, which will contain evidence from the past input, carried over the hidden states.

This architecture has the advantage of processing input of any length without increasing the model size. This makes RNNs suitable for natural language processing where the input is sequential. In practice, however, vanilla RNNs fail to learn long-distance dependencies, due to the vanishing/exploding gradients [3].

Long Short-Term Memory Network

It is a special variant of Recurrent Neural Networks, capable of learning long-distance dependencies [21]. LSTMs are able to remember information for long periods of time unlike simple RNNs, as they overcome the problems of vanishing and exploding gradients present in the traditional RNNs. They can process sequence of data as well as single data points. Therefore, they are suitable for processing images, text, speech or videos.

A typical LSTM unit has a cell, an input gate, an output gate and a forget gate. The cell is the unit where the information is stored and the gates control the flow of information into and outside the cell [21]. The long-term dependencies are stored within the cell and are referred to as the cell state. The input, forget and output gates regulate the amount of information that flows into the cell, the amount of information that remains in the cell and the amount of information that is used to compute the activation of the LSTM unit, respectively. They consist of a sigmoid neural net layer and an element-wise multiplication operation. The sigmoid layer outputs values between 0 and 1, describing the amount of information that should go through. A value of 0 means no information should go through while 1 means all the information must go through.

They are used for tasks like classifying, processing and making predictions based on time series data.

Colah's Blog[2] explains how LSTMs work in a simplified way. I give a brief description of the steps, illustrated with pictures taken from his blog.

Firstly, LSTM decides what information to forget from the cell state C_{t-1}. This decision is made by the forget gate layer, which looks at current input x_t and previous hidden state vector h_{t-1} and outputs a number between 0 and 1 for each number in the cell state C_{t-1}, using a sigmoid function. 0 means that the number will be completely forgotten and 1 means that it will be completely remembered. (Figure 2.6a)

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

The next step is to decide what new information will be stored in the cell state C_t. This consists of two steps. First, the sigmoid function of the input gate decides which values will be updated. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. (Figure 2.6b)

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C)$$

In the next step, the cell state is updated, by combining the old one C_{t-1} with the new one $\tilde{C}_t$. By multiplying the old state by f_t, LSTM forgets things that need to be forgotten. Then $i_t\tilde{C}_t$ is added. This represents the new candidate values, scaled by how much each state value is updated. (Figure 2.6c)

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

where $\odot$ denotes element-wise multiplication.

Finally, the output will be based on the cell state C_t, but will be a filtered version. First, the sigmoid function of the input gate decides what parts of the cell state will serve as output. Then, the cell state goes through a tanh layer (so that the values are between -1 and 1) and it is multiplied by the output of the sigmoid gate, to output the desired parts only. (Figure 2.6d)

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot tanh(C_t)$$

[2]Christopher Olah, Understanding LSTM Networks, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

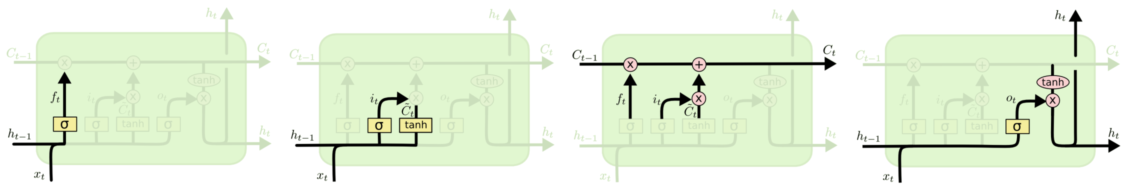


Figure 2.6: How an LSTM unit works, explained in 4 steps. a) Forget gate layer decides what information to forget from the old cell state. b) Input gate layer decides the new information that will be stored in the cell state. c) Cell state is updated. d) Output gate layer decides what to output. Images are taken from Colah's blog.

Bidirectional LSTM Network

LSTMs only preserve information of the past because inputs from the past are what they see. However, in sequence labeling tasks, both past and future inputs for a given time are known. In order to capture information from both directions, bidirectional LSTMs (BiLSTMs) were introduced [18]. Their effectiveness for numerous NLP tasks, NER included, has been proven by previous work [12].

In BiLSTMs, input will be processed in two directions, from left to right and the other way around. The additional backward LSTM preserves information from the future and by combining the hidden states from both LSTMs, information from both past and future in any point in time can be preserved. The structure of a typical BiLSTM layer with 3 units is given in Figure 2.7.

Suppose the next word in a sentence is to be predicted. This is what a unidirectional LSTM will see on a high level: **Er spricht ...**

It will try to predict the next word only by taking this context into consideration. On the other side, a BiLSTM has access to the future inputs, so it will be able to see further information for example:

Forward LSTM: **Er spricht ...**

Backward LSTM: **... , weil er in Frankreich wohnt.**

Using the information from the future makes it easier for the model to guess that the most probable next word is "Französisch".

Language Models

Neural networks can be used for language modeling, that is, estimating the probability distribution over sequences of words in order to predict the next word in a sequence. Their internal hidden layers extract features from the raw data and build internal representations for each word in the input sequence. The same word is mapped to different vector representations, depending on its position and the context it is used in the sentence. Such language-model-augmented knowledge has enhanced the quality of embeddings and empirically improved the performance of sequence labeling tasks, including NER [11][8][2].

Recent language models based on RNNs model languages as distributions over sequences of characters, instead of words [24]. They are called **character-level language models**. This approach has been successful at modeling languages with rich morphology (Arabic, German,

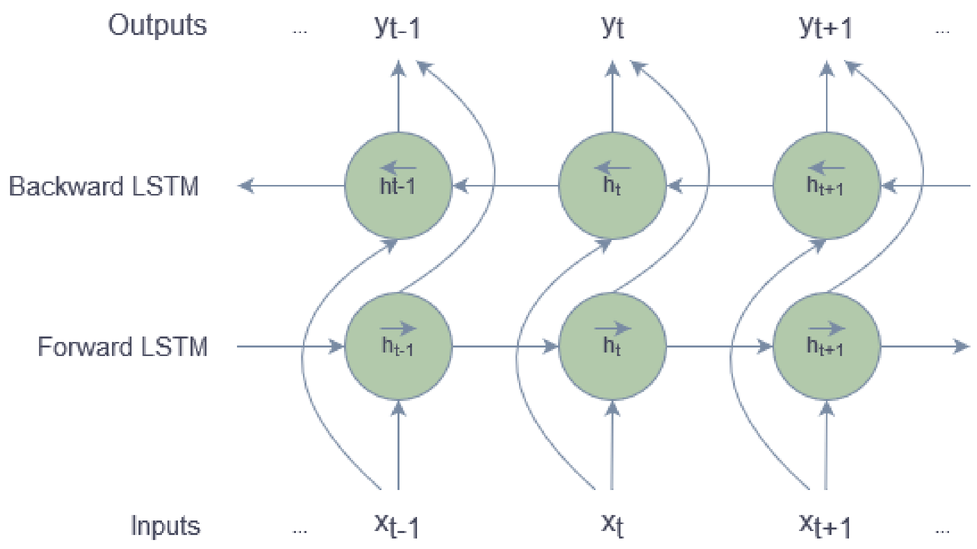


Figure 2.7: Bi-LSTM

Czech, Russian) [24]. Akbik et al. [2] followed a similar approach, using a BiLSTM network as a language modeling architecture, which takes as input a sequence of characters and predicts the next character. The hidden states of a bidirectional recurrent neural network are utilized to create contextualized word embeddings. Alongside with the forward language model, a backward model works in the same way, from right to left. The former model produces hidden state representations that encode semantic-syntactic information of the sentence up to the word's last character, including the word itself. Similarly the latter model produces hidden state representations that encode semantic-syntactic information from the end of the sentence to the word's first character. Both these vectors are concatenated to create the final embedding, which captures the semantic-syntactic information of the word and its surrounding text.

This approach handles rare and misspelled words better and models subword information as well.

Transformers

RNNs deal with sequential input by processing one word at a time. Recent architectures, transformers can parallelize the input, by processing the whole sequence simultaneously. They make better use of GPUs, designed for parallel computation, and accelerate the training process. Transformers have improved on the results for many common NLP tasks, including translation and constituency parsing [47]. Figure 2.8 illustrates the structure of a transformer, followed by a high-level description of how it works.

A transformer receives a sequence of words as input, each mapped to its embedding. The same word in different sentences might have different meanings. Transformers use a positional encoder to produce a vector that gives context based on the position of a word in the sentence. Combining both vectors results in another vector with positional information, which is passed into the encoder block. Firstly, the sequence goes through an self-attention layer that captures the contextual relationships between words in a sentence. For each word, the layer generates an attention vector, whose values indicate how relevant is the word w.r.t every other word in the sequence. Attention vectors are passed on to a feedforward network layer and transformed into

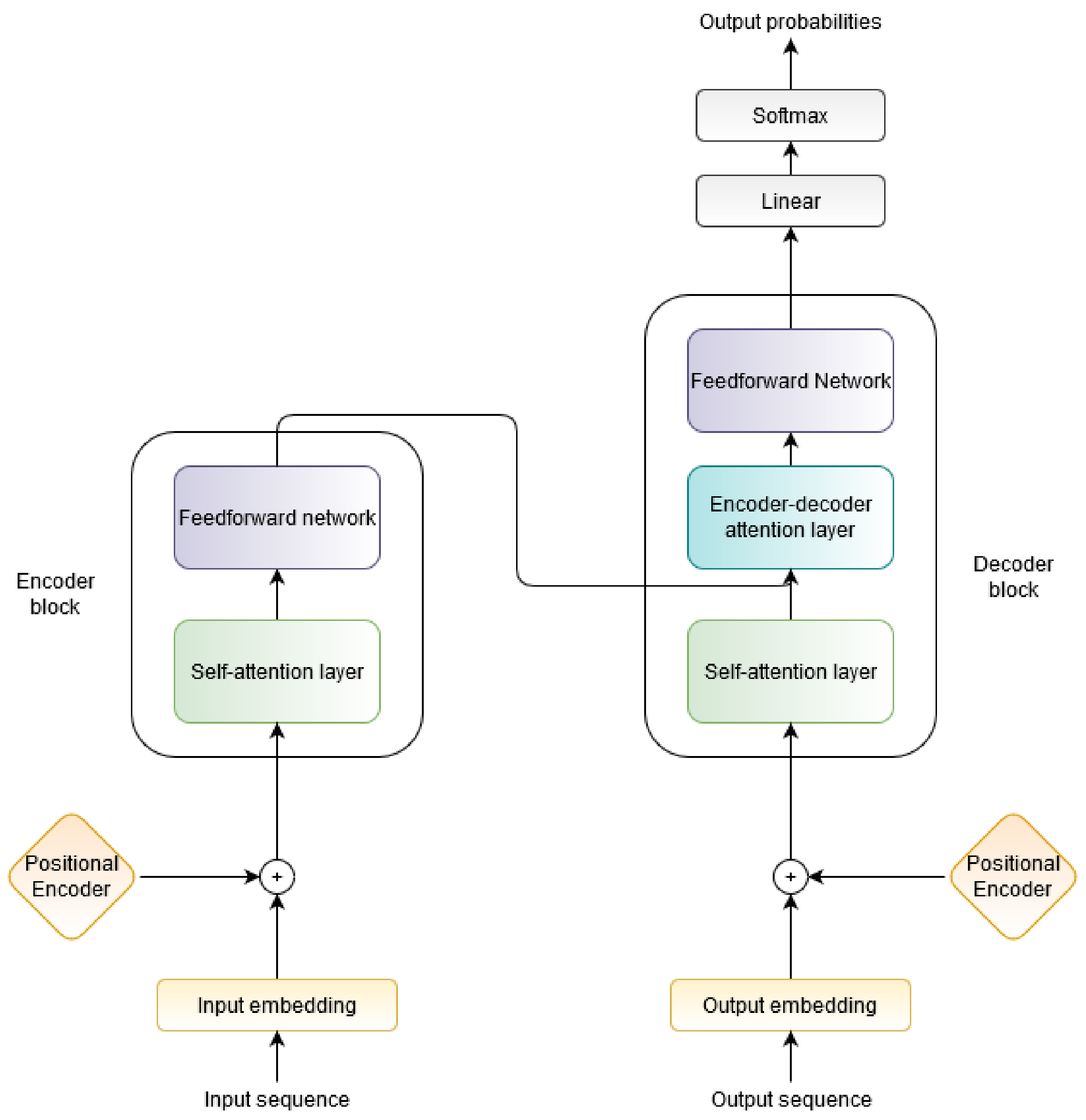


Figure 2.8: Transformer architecture

a form that is digestible by the next processing block.

During the training phase, the output sequence is fed to the decoder block, in a similar way as the input sequence is fed to the encoder. Again, the sequence goes through a self-attention layer and the resulting attention vectors are combined with the vectors from the encoder, to be processed by another attention block, called encoder-decoder attention layer. Here is where the mapping between both input and output sequences happens. The resulting vectors are then passed to a feedforward network layer layer, which makes the output vector more digestible for the next linear layer with as many hidden units as the length of output sequence. A softmax layer transforms its output into a probability distribution, based on which the next element of the sequence is predicted.

Two popular transformer architectures are:

BERT "**B**idirectional **E**ncoder **R**epresentations from **T**ransformers" is a transformer-based language model, developed by Google [11]. It corrupts input sentences by replacing some tokens with a [MASK] symbol. This is known as the masked language modeling (MLM) strategy. The model is then trained to reconstruct the original token. However, this method of training is somewhat restricted in that the model only learns from the masked out tokens which typically make up about 15% of the input tokens. BERT pre-trains deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. Its word representations are continuously informed by the words around them. BERT model uses several layers of transformer encoders and each layer produces an output for each word, which can be used as a word embedding. By feeding these embeddings as input features to a sequence tagger and observing the resulting F-scores, the BERT authors showed that concatenated embeddings from several layers yield better results. Regarding the combination strategy, the best performing choices for the task of NER are summing or concatenating the last 4 layers.

ELECTRA "**E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately" is another modern transformer-based language model that is trained to build representations that take into account the context around a specific word. Instead of masking out tokens like BERT, a portion of the input tokens are substituted by a synthetically generated token [9]. The model is then trained to classify whether each input token is original or substituted, thus allowing for gradient updates at every input position. This is achieved by two components that are trained jointly: a discriminator that detects the replaced tokens and a generator that provides plausible token substitutes.

2.3.3 Tag Decoders

Tag decoder is the final component of an NER model. It predicts tags for tokens in the input sentence. It receives the output of the context encoder and computes a distribution score for all possible tags for the words in the network input. Based on the distribution scores, it outputs a tag for each word in the input sentence.

Softmax

The entity recognition task is treated as a multi-class classification problem, when a softmax layer is used as a tag decoder. Tags are independently predicted based on the context-dependent representations, produced by the encoder, without taking into account its neighboring words. Softmax layer must have the same number of nodes as the number of tags. It takes as input the context-dependent representation for each word from the previous layer and assigns decimal probabilities of the current word having a particular tag, which must add up to 1.0. Softmax is defined as:

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_j x_j}$$

where x is the matrix of scores output by the BiLSTM network, with each row corresponding to a word in the input sequence. x_i is the current row, for which probabilities for each tag will be computed.

Conditional Random Field

Making independent tagging decisions for each output is limiting when there are strong dependencies across output labels. In NER, there are several hard constraints (e.g. I-PER cannot follow B-LOC) that cannot be modeled with independence assumptions. Conditional random fields jointly decode a chain of labels, ensuring the resulting label sequence to be meaningful. CRFs have been included in most state-of-the art models in NER, to avoid the generation of illegal annotations. Generally, they are built upon the encoder layer.

Suppose $x =< x_1, x_2, ..., x_n >$ is an input sequence where x_i is the vector for the i_{th} word. C denotes the matrix of scores output by the BiLSTM layer below. C has a size of $n \times k$, where k is the number of distinct tags, and $C_{i,j}$ corresponds to the score outputted by the network for the i^{th} word and the j^{th} tag. For a sequence of predictions, the score of a sequence of tags $y =< y_1, y_2, ..., y_n >$ is defined as:

$$s(x, y) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} C_{i, y_i}$$

where A is a tag-transition matrix such that $A_{i,j}$ represents the score jumping from tag i to tag j.

The score is normalized over all possible tag sequences using a softmax, yielding a probability for the sequence y:

$$p(y|x) = \frac{\exp(s(x, y))}{\sum_{\tilde{y} \in Y_s} \exp(s(x, \tilde{y}))}$$

During training, the log-probability of the correct tag sequence is maximized:

$$\log(p(y|x)) = s(x, y) - \log(\sum_{\tilde{y} \in Y_x} \exp(s(x, \tilde{y})))$$

where Y_x is the set of all possible tag sequences for the input sequence x. While decoding, the output sequence that obtains the maximum score given by:

$$y^* = \arg\max_{\tilde{y} \in Y_x} s(x, \tilde{y})$$

is chosen. At inference time, the last two equations can be computed using Viterbi algorithm.

2.4 Evaluation

There are two types of evaluation methods: the token-level method and the entity-level method. As the task name “Named-Entity” suggests, it is important to know how a model predicts the whole entity, instead of separate tokens. Therefore, entity-level method is the popular evaluation approach for NER models. It includes two steps: boundary detection and type identification. An instance is correctly recognized when the system correctly identifies its boundaries and type.

Numerically, the performance of such systems is measured by precision, recall and F-score. They are computed using the numbers of False Positives (FP), False Negatives (FN) and True Positives (TP).

- False Positive (FP): An entity found by the NER system that doesn’t exist in the ground truth.
- False Negative (FN): An entity not found by the NER system that exists in the ground truth.
- True positive (TP): An entity found by the NER system that also exists in the ground truth.

Precision takes into account only the number of correctly assigned labels (true positives).

$$Precision = \frac{TP}{(TP + FP)}$$

Recall, in addition to true positives, also takes into account the true tags that were missed, and therefore replaced with wrongly labeled ones.

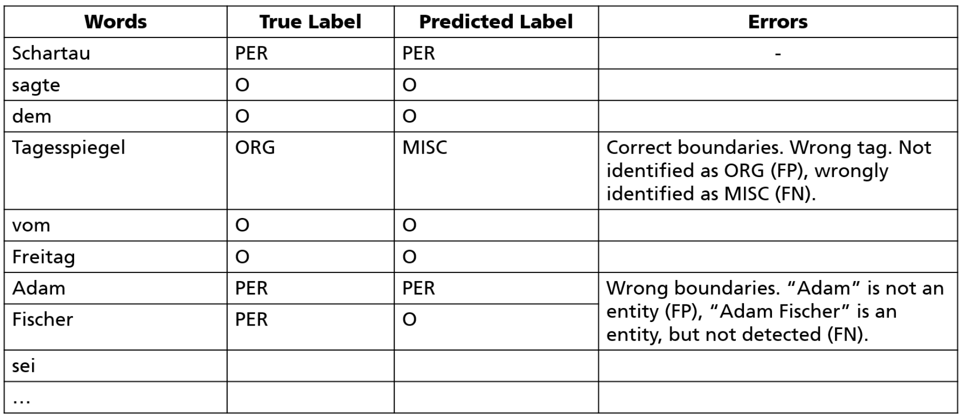
$$Recall = \frac{TP}{(TP + FN)}$$

F-score is a combination of precision and recall. Usually the traditional F-score, the harmonic mean of precision and recall, is used to measure the performance of NER systems. This score is very often referred to as F1-score too and it is commonly reported in percentage points (0-100%).

$$F_{score} = \frac{2 \cdot (Precision \cdot Recall)}{(Precision + Recall)}$$

Let’s consider the example illustrated in Figure 2.9. “Schartau” is counted as a TP because the true and predicted labels match. The model has failed to correctly identify "Tagesspiel". Instead of ORG, the system has labeled it as MISC. The entity is counted as two errors, a FN for ORG and a FP for MISC. When the model makes the boundary error, it is counted as two errors too. "Adam Fischer" hasn’t been entirely identified as a person, so it is counted as a FN for PER. On the other side, "Adam" detected by the model as a person, is not part of the true entities, so it will be counted as a FP for PER. In total, there are 1 TP, 2 FPs and 2 FNs. These values are used to measure precision, recall and F score, based on the above mentioned formulas:

$$Precision = \frac{1}{(1 + 2)} = 0.333$$



Words	True Label	Predicted Label	Errors
Schartau	PER	PER	-
sagte	O	O	
dem	O	O	
Tagesspiegel	ORG	MISC	Correct boundaries. Wrong tag. Not identified as ORG (FP), wrongly identified as MISC (FN).
vom	O	O	
Freitag	O	O	
Adam	PER	PER	Wrong boundaries. "Adam" is not an entity (FP), "Adam Fischer" is an entity, but not detected (FN).
Fischer	PER	O	
sei			
...			

Figure 2.9: Example of a NER prediction

$$Recall = \frac{1}{(1+2)} = 0.333$$

$$F_{score} = \frac{2 \cdot (0.333 \cdot 0.333)}{(0.333 + 0.333)} = 0.333 = 33.3\%$$

2.5 German NER

2.5.1 German NER datasets

There are two main German datasets for the NER task. They are commonly referred to by the name of the task they have been designed for, namely CoNLL-03 and GermEval. The research community has been using them to investigate new NER approaches or improve the state-of-the-art.

CoNLL-03

The CoNLL-2003 shared task[3] data files consist of four columns separated by a space character. Each line contains a word and its three characteristics given as tags, namely a part-of-speech (POS) tag, a syntactic chunk tag and the named entity tag. Each sentence is followed by an empty line.

Sentences in the dataset are represented using the IOB1[4] tagging scheme (Inside, Outside and Beginning). Words tagged with O are outside of named entities and the I-XXX tag is used for words inside a named entity of type XXX. Whenever two entities of type XXX are immediately next to each other, the first word of the second entity will be tagged B-XXX in order to show that it starts another entity.

The sentences contain entities of four types: person (PER), organization (ORG), location (LOC) and miscellaneous (MISC) names that do not belong to any of the previous three types.

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

[4]There are two IOB tagging scheme types: IOB1 and IOB2. The difference is how they deal with how two adjacent named entities of the same type are labeled. In IOB2, all entities begin with B-. In IOB1, B- is only used to separate two adjacent entities of the same type.

The entities are assumed to be non-recursive and non-overlapping. When an entity is embedded in another one, usually only the top level entity has been annotated.

The CoNLL-2003 data for German is a collection of articles from the Frankfurter Rundschau, written in one week at the end of August 1992. People of the University of Antwerp have annotated the dataset.

GermEval

It is the largest publicly available dataset for German NER. It consists of 31,000 manually annotated sentences (over 591,000 tokens), sampled sentence-wise from German Wikipedia and German online news, using corpora from Leipzig Corpora Collection [39] as a basis.

The data files consist of sentences, extracted from different articles. Each sentence starts with a line containing the source link and the date of the article. Each word is put on a separate line and there is an empty line after each sentence. Words are followed by two annotations: outer and inner span named entities. For example, the term Bayern München is tagged as organization in the outer span annotation. However, the terms are annotated as location in the inner span annotation. In addition to the standard tags used in the CoNLL-03 dataset, more detailed versions of these entities were formed by adding suffixes: -deriv to mark the derivation of named entities (e.g. die deutsche Flugsicherung - deutsche is a derived location) and -part to mark compounds including an entity (e.g. in Karlsgymnasium - the part ‘Karl’ is a person). To be consistent with the tagsets of the CoNLL-03 task, the detailed entity types are removed for the NER task. However, these types can not be ignored for semantic tasks, such as identifying locations in articles.

The dataset was annotated by native German speakers, unlike the CoNLL-03 German dataset, making it more consistent. It follows the IOB2 annotation scheme. Another advantage of this dataset is that it is easy to obtain and use with no licensing issues of previous datasets[5].

2.5.2 Related work in German NER

There is already a number of empirical studies that investigate the performance of NER taggers on German texts. Since 2005, Stanford NER has provided CRF-based NER models for multiple languages, German included. Besides language-independent features, including character n-grams, words, word shapes, POS and lemmas, they have complemented the German model with distributional clusters, computed on a large German corpus [15]. The distributed model for German is pre-trained on the German CoNLL-03 Shared-Task data.

Another CRF-based NER system was developed later by Benikova et al. [5]. This model is optimized for the GermEval14 NER challenge and uses similar features to Stanford NER’s. GermaNER is supplemented by some specific information sources, such as distributional semantics and topic cluster information, gazetteer lists.

Over the past few years, neural architectures for NER that use no language-specific features resources or features have been developed. They can be easily adopted to any language, as long as there is supervised training data available for that language. The German dataset of CoNLL-03 Shared-Task has been used to evaluate the performance of several NER models for German.

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

Lample et al. [25] proposed a a bidirectional LSTM with a sequential conditional random layer above it (LSTM-CRF), that combined skip-n-gram embeddings with character-based word representations generated from a Bi-LSTM model taking a sequence of characters as input. Experiments in German showed that BiLSTM-CRF was able to obtain state-of-the-art NER performance without any hand-engineered features or gazetteers.

Another LSTM-based model was proposed later by Gillick et al. [17]. Byte-to-Span (BTS) read text as bytes and outputs span annotations of the form [start, length, label] where start positions, lengths, and labels were separate entries in the vocabulary. Due to the small vocabulary size, BTS models were very compact, but produced results similar to the state-of-the-art in POS-tagging and NER for German.

In 2018, Yadav et al. [49] extended Lample's NER approach with a learned representation of the n-gram prefixes and suffixes of each word. Results showed that sub-word features such as prefixes and suffixes are complementary to character and word-level information, slightly improving the state-of-the-art performance for German.

Riedl & Pano [40] came to the conclusion that Bi-LSTMs profit substantially from transfer learning, which enables them to be trained on multiple corpora, resulting in a new state-of-the art model for German NER. They pitted linear chain CRFs against BiLSTMs, to observe the trade-off between expressiveness and data requirements and found out that neural networks outperform CRFs, when enough data is available.

In the span of little more than 3 years, transfer learning in the form of pretrained language models has become ubiquitous in NLP and has contributed to the state-of-the-art on a wide range of tasks, including NER. The general practice is to pretrain representations on a large unlabeled text corpus using a method of choice and then to adapt these representations to a supervised target task using labeled data.

Devlin et al. [11] released mBERT (Multilingual BERT), pre-trained on the concatenation of monolingual Wikipedia corpora from 104 languages, including German. Akbik et al. [2] introduced the contextualized string embeddings and outperformed previous work on German NER, reporting new state-of-the-art scores on both the CoNLL-03 Shared Task and GermEval challenge data.

Schweter and Baiter [41] investigated the influence of using pre-trained language models for named entity recognition for "low-resource" historical german datasets. Their work showed that this approach was more effective than using transfer-learning (Riedl & Pado).

The most recent contribution in German-specific language models has been made by Chan et al. [8]. They used a range of different German language corpora such as OSCAR (Common Crawl), OPUS (movie subtitles, parliament speeches, books), Wikipedia, OpenLegalData (German Court Decisions) for pre-training BERT and ELECTRA models. By benchmarking against existing German models for several NLP tasks, these models have proved to be the best German models to date.

Table 2.1: Performance scores reported by each paper for the NER task in German.

Paper/Dataset	CoNLL-03	GermEval
Stanford NER	78.20	-
GermaNER	79.37	76.52
Lample et al.	78.76	-
Gillick et al.	76.22	-
Yadav et al.	79.01	-
Riedl & Pano	82.93	84.73
Akbik et al.	88.32	84.65
Chan et al.	-	88.95

Chapter 3

Comparative evaluation of embeddings for German NER

One of the variables of interest is the type of embeddings. Some embeddings might be more useful than others for German NER. This chapter describes the process of conducting a comparative evaluation, in order to analyze how different embeddings compare to each other for the two biggest NER datasets in German, CoNLL-03 and GermEval. Embeddings are selected in such a way, that every category described earlier in Chapter 2 is represented by at least one of them. They are fed as input to a popular DL-based sequence labeling architecture. Weaknesses of each embedding type are exposed through a detailed analysis of their performance w.r.t a set of dataset attributes.

3.1 Chosen architecture for evaluation

Out of all the deep learning sequence labeling architectures proposed for NER, bidirectional LSTM networks with a CRF layer on top of it have achieved state-of-the-art performance (Ma and Hovy [28], Lample et al. [25]).

Words in an input sequence are mapped to embeddings and are sequentially processed by a BiLSTM network. Both forward and backward LSTM units compute representations of the left and right context of the sentence at every word. Both vectors are concatenated into a final context-dependent embedding. The CRF layer ensures that the predicted labels are legal, by jointly modeling the probability from the Bi-LSTM layer and the transition probability between adjacent tags. Tag transition probabilities can be learned through the CRF layer during the training process.

The architecture will be referred to as the BiLSTM-CRF module.

3.2 Evaluation

In order to evaluate the models, the *conlleval* script from CoNLL-03 Shared-Task is used. It computes the precision, recall and F-score for each entity type and combined. The combined scores are reported and compared to find the best performing models.

On the other side, holistic metrics like F-score do not provide insight into how particular models perform differently and how diverse datasets impact the model design choices. Fu et al. [16] introduced a fine grained evaluation methodology for NER that interprets the differences in models and datasets by exposing the strengths and weaknesses of models. This evaluation approach leverages the notion of “attributes” defined over tokens/entities[1] of a dataset. The attributes can be classified as local or aggregate.

Local attributes characterize properties of a token or an entity, regarding the token/entity itself or the sentence in which they appear. Local attributes used in this work are:

- **Entity Length (eLen)**: The number of tokens an entity spans over.
- **Sentence Length (sLen)**: The number of tokens in a sentence.
- **Entity Density (eDen)**: How often an entity word appears in a sentence in relation to the total number of tokens it contains.

Aggregate attributes characterize properties of entities or tokens based on aggregate statistics that require computations over the whole training data. Aggregate attributes used in this work are:

- **Label Consistency of a Token (tCon)**: How consistently a particular token is labeled with a particular label.
- **Token Frequency (tFre)**: The number of times a token appears in a dataset over the total number of tokens.
- **Label consistency of an Entity (eCon)**: How consistently a particular entity is labeled with a particular label.
- **Entity Frequency (eFre)**: The number of times an entity appears in a dataset over the total number of entities.

Test entities are partitioned into a set of buckets for each pre-defined attribute. Comparing the F-scores among different buckets helps us interpret the differences in models and datasets, as well as the relationship between them. Figure 3.1 illustrates the process in a simplified way.

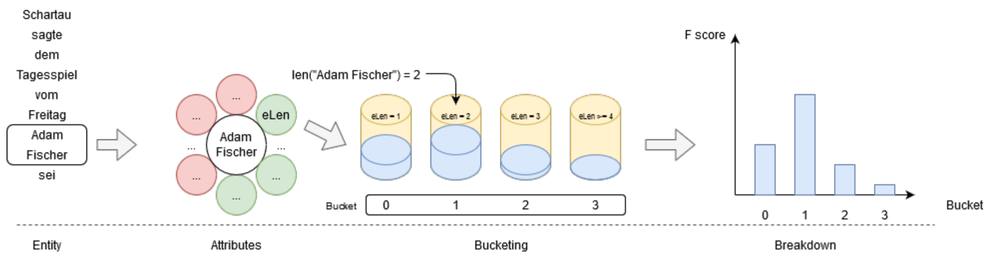


Figure 3.1: An example of the evaluation methodology. *eLen* (entity length) is one of the attributes of the entity "New York". Performance can be droken down over the defined attribute values.

[1]A token is a single word, it can either be an entity, a part of an entity or not an entity at all. An entity can span one or more words.

3.3 Datasets

CoNLL-03 and GermEval consist of 3 sets: a training set, a test set and a development set. The training set contains the largest portion of the dataset and is used to train a neural network, the development set can be used to finetune the parameters of network, while the test set is used to evaluate the performance of a network after training. Table 3.1 shows the number of sentences for the three sets for each dataset and Table 3.2 shows the number of entity instances for each set on both datasets.

Table 3.1: Number of sentences for each set

Dataset	Train	Dev	Test
CoNLL-03	12152	2867	3005
GermEval	24000	2200	5100

Table 3.2: Dataset statistics

	CoNLL-03			GermEval		
Entities	Train	Dev	Test	Train	Dev	Test
PERSON	4495	1990	1824	12170	1152	2551
ORGANIZATION	4242	2117	1262	9025	862	1848
LOCATION	5239	1340	1285	9450	907	2009
MISC	2807	1141	800	6531	589	1563

3.3.1 Dataset pre-processing

The datasets are brought to the same column-oriented format as CoNLL-03, where the first column contains the tokens and the second one contains the NER tags. In CoNLL-03 dataset, the POS-tags and syntactic chunks were removed, as they are redundant. In GermEval, the inner span annotations were removed, in order to be consistent with the tagsets of CoNLL-03. However, they can not be ignored for semantic tasks, such as identifying locations in articles. The IOB2 annotation scheme is used for both datasets.

Building end-to-end systems is the goal, therefore no further dataset pre-processing is performed.

3.4 Chosen embeddings

FastText embeddings They are generated from a pre-trained context-independent model, an extension of the skip-gram model [7], which takes into account subword information. The approach models morphology by considering character n-grams and each word is represented as a bag of character n-grams. This is convenient for German, as it is a morphologically rich language.

Character-level embeddings They are not pre-trained, as they depend on the task. They get randomly initialized at first, so they are not meaningful until trained on the specific NER task. Stacking them with pre-trained embeddings, one can obtain higher-quality word representations. They are generated following Lample's approach [25]. A character-level BiLSTM layer is applied to each word separately and the concatenated hidden output states represent the embeddings.

Flair embeddings They are extracted from the internal states of a pre-trained character-level language model [2]. Section 2.3.2 describes in detail how Flair embeddings are produced.

Transformer-based embeddings Transformers used in this work are multilingual BERT, GBERT and GELECTRA. They are uploaded to the Hugging Face model hub[2] as *bert-base-multilingual-cased*, *deepset/gbert-large*, *deepset/gelectra-large*. The top four layers of these transformers are exploited to create the contextualized embeddings for the NER models.

The dimensions for each type of embeddings can be found in Table 3.3.

Table 3.3: Types of embeddings used and their dimensions

Embeddings	Dimension
FastText	300
Character-level	50
Flair	8192
mBERT	3072
GBERT	4096
GELECTRA	4096

3.5 Experiment

3.5.1 Experimental setup

The chosen word embeddings are passed into a BiLSTM-CRF module, whose structure is shown in Figure 3.2.

The model setups used for evaluation are as follows:

$BASE$: This is the simplest setup, as it relies only on pre-trained FastText embeddings. It is a reimplementation of the LSTM-based model of Huang et al. [22].

$BASE_{+CHAR}$: An extension of $BASE$, in which character-level embeddings are added to FastText embeddings, after being computed by a BiLSTM layer for each word, they . It is a reimplementation of Lample's neural architecture for NER [25].

[2]https://huggingface.co/models

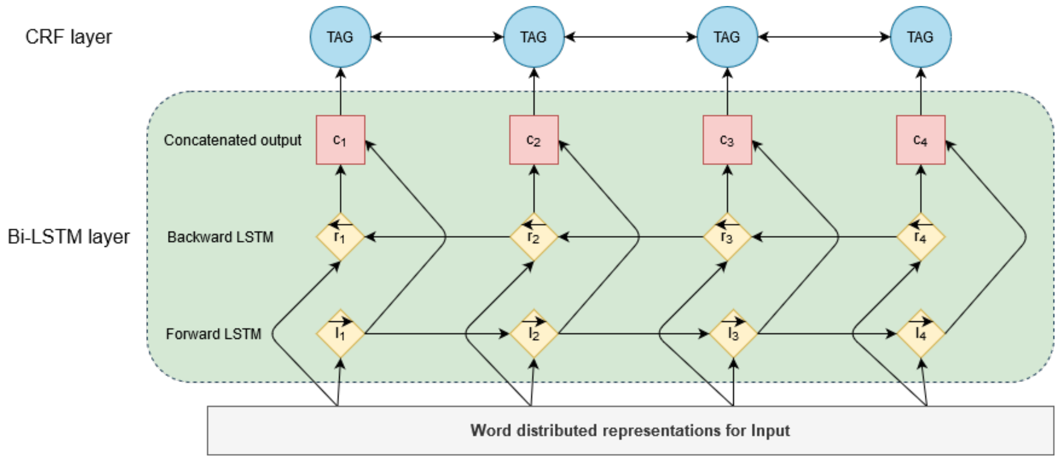


Figure 3.2: BiLSTM-CRF module

$BASE_{+FLAIR}$: A similar extension, in which FastText embeddings are concatenated with Flair embeddings.

$BASE_{+mBERT}$: In this setting, embeddings extracted from multilingual BERT (mBERT) are concatenated with FastText embeddings.

$BASE_{+GBERT}$: A setup, in which embeddings extracted from GBERT and pre-trained FastText embeddings are concatenated and passed to the BiLSTM-CRF module.

$BASE_{+GELECTRA}$: This model uses the contextualized embeddings from GELECTRA and concatenates them with the pre-trained FastText embeddings.

The first two setups, $BASE$ and $BASE_{+CHAR}$, are shown in Figures 3.3 and 3.4, respectively. The last four setups share the same structure, they simply use different language models to produce their embeddings. Figure 3.5 illustrates the structure, labeled as $BASE_{+LM}$, where LM can be any of the four language models, mentioned above.

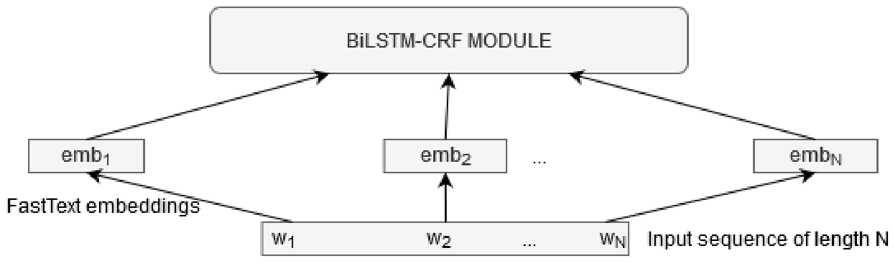


Figure 3.3: BASE

3.5.2 Model characteristics and training parameters

Language Models: Figure 3.6 below gives an overview of the language models being used (mBERT, GBERT, GELECTRA and FLAIR) and their characteristics.

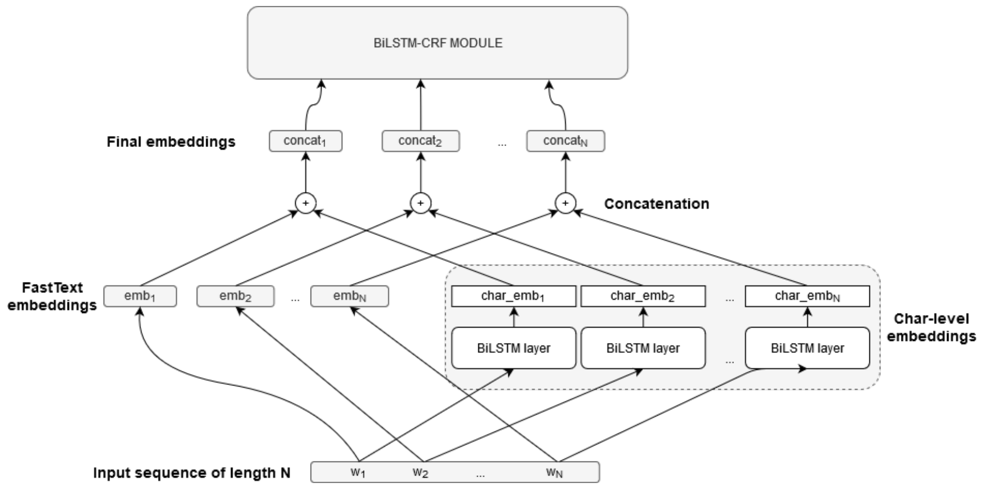


Figure 3.4: BASE$_{+CHAR}$

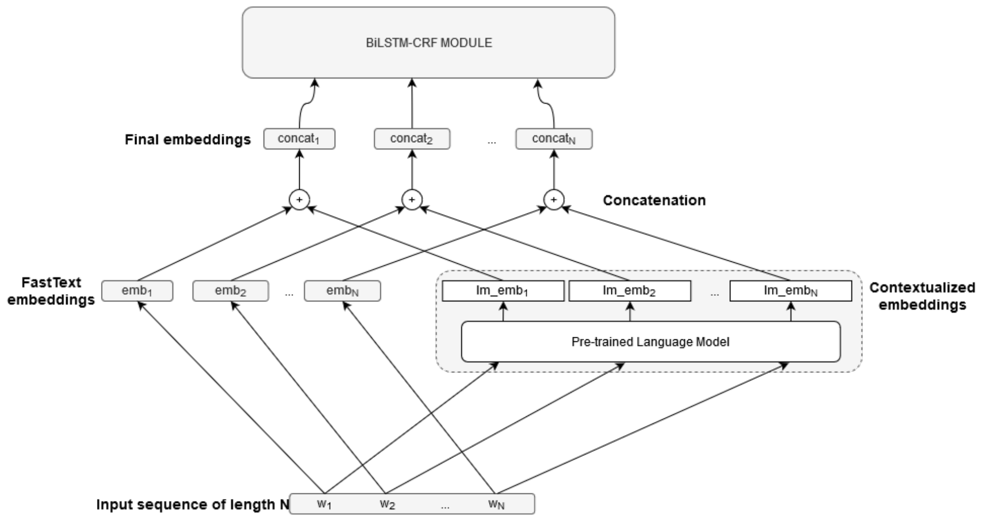
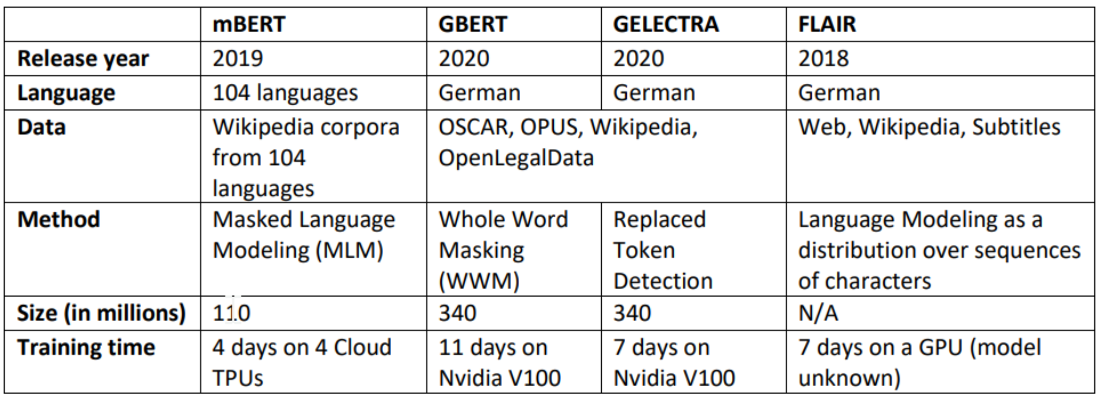


Figure 3.5: BASE$_{+LM}$



	mBERT	**GBERT**	**GELECTRA**	**FLAIR**
Release year	2019	2020	2020	2018
Language	104 languages	German	German	German
Data	Wikipedia corpora from 104 languages	OSCAR, OPUS, Wikipedia, OpenLegalData		Web, Wikipedia, Subtitles
Method	Masked Language Modeling (MLM)	Whole Word Masking (WWM)	Replaced Token Detection	Language Modeling as a distribution over sequences of characters
Size (in millions)	110	340	340	N/A
Training time	4 days on 4 Cloud TPUs	11 days on Nvidia V100	7 days on Nvidia V100	7 days on a GPU (model unknown)

Figure 3.6: Characteristics of mBERT, GBERT, GELECTRA and FLAIR language model.

BiLSTM-CRF module: I follow most hyper-parameter suggestions as given by the thorough study from Reimers and Gurevych [38]. All model setups share a generic SGD forward and backward training procedure.

I perform model selection over the learning rate $lr \in \{0.05, 0.1, 0.15\}$ and batch size $\in \{16, 32, 48\}$, in order to obtain the values for which the model performs the best on the validation set. A batch size of 32 sentences was the most effective choice for training and updating parameters of the models. I employ a learning rate of 0.1 that is halved if training loss does not decrease for 5 consecutive epochs. The gradient clipping is set to 5.

The number of hidden states for the BiLSTM layer is set to 512 and each of them is randomly initialized with values drawn from the standard normal distribution. For the $BASE_{+CHAR}$ setup, a BiLSTM layer with 25 hidden states is used to train the character-level embeddings.

Each model is trained for 150 epochs. The experiment is repeated three times and the best F-score on the test set is reported as final performance.

3.5.3 Implementation, Training and Tagging Speed

The neural networks are implemented using the open-source Flair[3] library, built directly on PyTorch. It is the most convenient framework to use, as it is well-documented and provides the necessary embeddings.

In a Quadro RTX 4000 processor, training time varies from 3 hours for $BASE$ to 9 hours for $BASE_{+FLAIR}$, $BASE_{+mBERT}$, $BASE_{+GBERT}$ and $BASE_{+GELECTRA}$. $BASE_{+FLAIR}$ takes the longest to train. The rest take a few hours less, approximately up to 5 hours. Tagging the CoNLL-03 test set takes about 26 seconds, while for GermEval it takes around 49 seconds.

3.6 Results

3.6.1 Comparative evaluation

The experimental results are summarized in Table 3.4.

Table 3.4: Summary of evaluation results or all proposed setups.

	CoNLL-03			GermEval		
Model	Precision	Recall	F-score	Precision	Recall	F-score
BASE	78.35	71.45	74.74	85.02	76.75	80.68
$BASE_{+CHAR}$	82.39	77.56	79.9	85.91	80.89	83.33
$BASE_{+FLAIR}$	86.13	80.97	83.47	87.04	84.11	85.55
$BASE_{+mBERT}$	86.01	80.78	83.32	89.28	85.98	87.6
$BASE_{+GBERT}$	**86.89**	**82.09**	**84.42**	88.56	85.3	86.9
$BASE_{+GELECTRA}$	86.24	81.66	83.89	**90**	**87.54**	**88.75**

Evaluating the same network architecture, while varying the embeddings for input makes it easier to quantify their impact on the NER performance. Looking at the F-score column

[3]Flair, https://github.com/flairNLP/flair

for both datasets, the scores improve as FastText embeddings ($BASE$) are complemented with character-level embeddings ($BASE_{+CHAR}$) or contextualized embeddings from FLAIR, mBERT, GBERT or GELECTRA. The best performance is achieved when transformer-based embeddings are concatenated with FastText. $BASE_{+GBERT}$ and $BASE_{+GELECTRA}$ are the winning models on CoNLL-03 and GermEval, respectively. A common characteristic of GBERT and GELECTRA is that they are both transformer-based language models, trained on a large German corpora.

Detailed observations are listed:

1. **Better performance on GermEval than CoNLL-03**. The most striking observation is that each model performs better when it is trained and tested on GermEval data, compared to CoNLL-03. Better precision, recall and F-scores indicate more accurate entity extractions. According to Benikova et al. [4], *GermEval is a higher quality dataset*, as it has been annotated by native speakers and the annotation process has gone through several stages, in order to mitigate the consistency issues, present in CoNLL-03. Consistency, as it will be shown later, can help systems make more accurate tagging decisions and perform better at entity extraction.

2. **Character-level embeddings make a difference in German NER**. There is an increase in the F-scores for both datasets, when switching from $BASE$ to $BASE_{+CHAR}$. Task-trained character-level embeddings ($BASE_{+CHAR}$) cause an improvement with 5.16 points on CoNLL-03 and 2.16 points on GermEval data. As discussed earlier, they are useful for handling out-of-vocabulary words. German is a morphologically rich language, with many rare compound words. The task-trained embeddings encode information specific to the task and domain. This impacts the performance scores on both datasets.

3. **Transformer-based embeddings achieve the best performance.** Drawing conclusions from the comparison of Flair, mBERT, GBERT and GELECTRA is not trivial because they differ in dimensions, language modeling strategy and pretraining data. Flair underperforms GBERT and GELECTRA on both datasets. Compared to multilingual BERT, Flair performs slightly better on CoNLL-03, but worse on GermEval. This might indicate that character-level contextualization is less useful than word-level contextualization for German. Regarding transformers only, $BASE_{+GBERT}$ is the winner on CoNLL-03 dataset (84.42) and $BASE_{+GELECTRA}$ on GermEval (88.75). Both of the models are regarded as "the best German models up to date" [8] and the claim is supported by the comparative evaluation results in this work.

3.6.2 Fine-grained evaluation

In order to further explain the results from Table 3.4, the fine-grained approach described earlier is used. The code, provided on Github[4] from the authors of the approach, does not support German datasets, so it is extended for German CoNLL-03 and GermEval, according to the method described in their paper [16]. Two types of analysis were performed: model-wise and bucket-wise.

[4]https://github.com/neulab/InterpretEval

Model-wise analysis

Model-wise analysis is used to gain insights into how different dataset attributes affect the performance of models with different embedding configurations.

Approach – The test data from each dataset, CoNLL-03 and GermEval, is split into 4 buckets for each of the specified data attributes. The bucketing interval strategy with respect to each attribute is described in the Appendix.

Statistical variables $S^{\sigma}_{model,attrib.}$ and $S^{\rho}_{model,attrib.}$ are used to draw correlations between an attribute and a model's performance.

- $S^{\rho}_{model,attrib.}$ calculates the Spearman's rank correlation coefficient for a specific model, using the ranked F-scores of buckets for an attribute. It characterizes the strength of the relationship between the values of an attribute (buckets) and a model's performance.
- $S^{\sigma}_{model,attrib.}$ calculates the standard deviation among the F-scores achieved by a model in each bucket of an attribute. Intuitively, it characterizes the degree to which an attribute impacts the performance of a model.

For example, $S^{\rho}_{BASE_{+FLAIR},eLen} = -0.9$ indicates that the performance of the model using Flair embeddings is negatively and highly correlated with the values of eLen, that is, the performance degrades as it deals with longer entities. Furthermore, $S^{\sigma}_{BASE_{+FLAIR},eLen}$ indicates the degree to which entity length influences the performance.

Table 3.5 illustrates the averages of $S^{\sigma}_{model,attrib.}$ and $S^{\rho}_{model,attrib.}$ on both CoNLL-03 and GermEval datasets. The most significant measures have been colored and their interpretations lead to the following observations:

Table 3.5: Model-wise measures, Spearman coefficient and standard deviation.

	Spearman (S^{ρ})							Standard deviation (S^{σ})						
Model	eCon	tCon	eFre	tFre	eLen	sLen	eDen	eCon	tCon	eFre	tFre	eLen	sLen	eDen
BASE	0.9	1	0.9	-0.2	-0.9	-0.1	0	0.0997	0.1233	0.0842	0.0703	0.1449	0.0257	0.0341
BASE$_{+CHAR}$	0.9	0.9	0.9	-0.2	-0.9	-0.2	0	0.0883	0.1073	0.0660	0.0651	0.1526	0.0207	0.0285
BASE$_{+FLAIR}$	0.9	0.9	0.9	-0.2	-0.9	-0.2	0	0.0770	0.0935	0.0535	0.0501	0.1402	0.0156	0.0263
BASE$_{+mBERT}$	0.8	1	1	-0.3	-0.8	-0.2	0.2	0.0675	0.0812	0.0457	0.0372	0.1204	0.0094	0.0200
BASE$_{+GBERT}$	0.8	1	1	-0.2	-0.8	0	0.3	0.0711	0.0805	0.0433	0.0415	0.1176	0.0169	0.0191
BASE$_{+GELECTRA}$	0.8	1	1	-0.3	-0.9	-0.5	-0.3	0.0704	0.0808	0.0361	0.0314	0.1006	0.0147	0.0206

1. **Entity length is negatively and strongly correlated with models' performances.** - High negative values of S^{ρ} for eLen suggest that the performances are sensitive to entity length. Models are less likely to detect entities that span over multiple tokens. The standard deviation values for models with transformer-based (mBERT, GBERT, GELECTRA) word embeddings are the lowest, compared to models using Flair embeddings or non-contextualized embeddings, making them the most robust models of the set. This might come as a result of the attention layers transformer-based LMs use, which give them the capability to model longer spans of entities better. $BASE_{+GELECTRA}$ has the smallest standard deviation for eLen, which makes it the most robust, when dealing with long entities.

2. **Label consistency is positively and highly correlated with the performance of NER models.** - High values of S^{ρ} for eCon and tCon indicate that models are generally sensitive to the consistency of annotations. To make the difference between the models, S^{σ} values among models are compared. $BASE$ has the highest standard deviation, suggesting that FastText embeddings alone are not informative enough to deal with type ambiguities that result in the same entity appearing multiple times in the dataset but being tagged differently. $BASE_{+mBERT}$, $BASE_{+GERT}$ and $BASE_{+GELECTRA}$ have the smallest standard deviations, indicating that they generalize better to entities with low label consistencies. This can be attributed to the ability of their embeddings to model polysemy, by taking the context around the word into account. They are more useful for generalization than traditional static embeddings like FastText or character-level embeddings, where a word is mapped to a single representation, regardless of its context.

Bucket-wise analysis

Approach – Given a model and an evaluation attribute, the buckets containing the test samples that have achieved the highest and lowest performance are selected. The difference in F-scores suggests under which conditions a model performs well or not. Table A.1 illustrates the bucket-wise evaluation for $BASE_{+GELECTRA}$ on both datasets, in the form of column charts. Each column represents an attribute, whose label is given above. The labels along x-axis represent the bucket number of the attribute, on which the model has achieved its worst and best performance, respectively. The dark green bin shows the absolute difference between the buckets. For example, considering $BASE_{+GELECTRA}$ on CoNLL-03, the bucket with the lowest F-score for entity consistency (eCon) is bucket number 1 and the one with the highest F-score is bucket number 3. The difference between the two scores (20.73) is shown through the length of the bin. Other models follow similar patterns and are included in the appendix.

Observations – Large gaps in performance are observed generally for entity length, where the lowest performance is obtained on the last bucket with entities spanning 4 or more tokens (eLen: bucket 3). Significant differences can be observed for attributes related to label consistency (eCon, tCon) too. All models score the highest in the last bucket of entity/token consistency (eCon, tCon: bucket 3), which contains the most consistently-labeled entities. On the other hand, the worst performance was obtained in the first two buckets with low consistency (eCon, tCon: buckets 0 and 1). Results show that even though models with contextualized embeddings like $BASE_{+FLAIR}$, $BASE_{+mBERT}$, $BASE_{+GBERT}$ and $BASE_{+GELECTRA}$ are better at dealing with lengthy entities, they still suffer from a considerable drop in performance.

3.6.3 Contributing in German NER

The results of the comparative evaluation can be used to "fill the gaps" in the German NER research. Most types of embeddings are not evaluated on both datasets. Table 3.7 shows a summary of evaluation results of all proposed setups and the best published scores for each of them on CoNLL-03 and GermEval. For the setups that have been evaluated before, the difference between performance scores is given in parenthesis.

Table 3.6: Bucket-wise analysis for each model. Numbers from 0 to 3 represent the four buckets, from the smallest to the largest attribute values. The "number/number" pairs in the horizontal axis represent the buckets of a specific attribute on which a model achieved its **worst** and **best** performance, respectively. Green bins represent the difference between the best and worst performance.

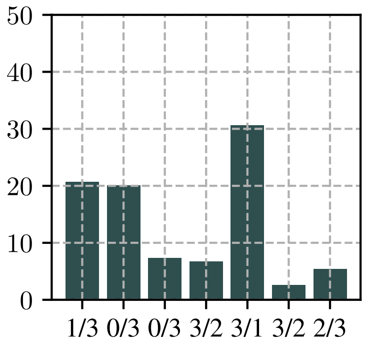
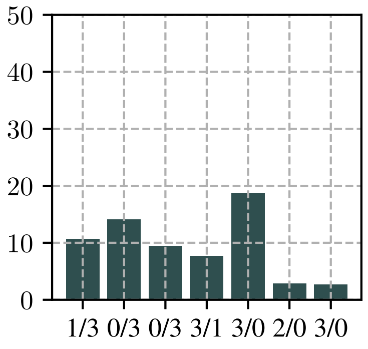
Model	CoNLL-03 (eCon, tCon, eFre, tFre, eLen, sLen, eDen)	GermEval (eCon, tCon, eFre, tFre, eLen, sLen, eDen)
BASE$_{+GELECTRA}$		

Table 3.7: Summary of evaluation results and the best published scores of all proposed setups on CoNLL-03 and GermEval.

Model	CoNLL-03 Published score	CoNLL-03 Our score	GermEval Published score	GermEval Our score
BASE	-	74.74	-	80.68
BASE$_{+CHAR}$	78.76	79.9 ($\uparrow$ 1.14)	-	83.33
BASE$_{+FLAIR}$	88.32	83.47 ($\downarrow$ 4.85)	84.65	85.55 ($\uparrow$ 0.9)
BASE$_{+mBERT}$	-	83.32	-	87.6
BASE$_{+GBERT}$	-	84.42	88.16	86.9 ($\downarrow$ 1.26)
BASE$_{+GELECTRA}$	-	83.39	88.95	88.75 ($\downarrow$ 0.2)

Authors of model setups $BASE$ [22] and $BASE_{+mBERT}$ [35] have not published scores for German NER on any of the datasets. Authors of $BASE_{+GBERT}$ and $BASE_{+GELECTRA}$ [8] have reported scores on GermEval dataset only, while $BASE_{+CHAR}$ [25] has been evaluated only on CoNLL-2003.

In the case of Flair embeddings on CoNLL-03, there is a big difference between the published and obtained score. This issue has been reported by other researchers, who have tried to reproduce the results for this setup [5]. This is caused by using different versions of the dataset. Authors of Flair use the revised version of this dataset (2006), whose statistics are different from the older version, used in this work (2003). Unable to obtain the revised dataset, this work reports the scores on the 2003 version.

[5]https://github.com/flairNLP/flair/issues/1102

Chapter 4

NER performance across other domain settings

The models that were trained on CoNLL-03 and GermEval in Chapter 3, can now be used to investigate other aspects of NER. In this chapter, I aim to improve the performance of NER models on a scarce-data domain and exploit the multilingual embeddings from pre-trained multilingual BERT to build a single model that extract entities in several languages.

4.1 Finetuning for a "small-data" scenario

A drawback of neural networks is that they are "data-hungry". They require a lot of training data, in order to properly adjust their weights and generalize well. In NER, the process of annotating a dataset requires a lot of time and efforts. As a result, the number of available datasets for NER is very limited. There are domains for which there is little to no available training data. The state-of-the-art models do not perform well on such domains, as they have not been covered by the data they have been trained on.

However, their knowledge on a domain might be transferred to the new domain with scarce data. In this experiment, I resume the training process of the existing models on a small dataset like Europarl. This technique is also known as **finetuning** and has been previously used by Riedl & Pado [40] to improve the NER results on historical German data. In their work, they used pre-trained FastText embeddings. Here, contextualized embeddings extracted from pre-trained language models are considered as well.

CoNLL-03 and GermEval data are considered as source domains, as they are big datasets.

Table 4.1: Europarl statistics

	Train	Test
PERSON	410	105
LOCATION	543	181
ORGANIZATION	730	144
MISC	662	304
Number of sentences	3538	857

The target domain, represented by Europarl, consists of parliament sessions and discussions in German. Statistics for this dataset can be found in Table 4.1. Since both source and target domains have the same label set (PER, ORG, LOC, MISC), all the pre-trained model parameters and layers are shared during the process of finetuning on the new data for another few epochs.

4.1.1 Experimental setup

The pre-trained models used for finetuning on Europarl are the setups from the first experiment, namely:

- $BASE$
- $BASE_{+CHAR}$
- $BASE_{+FLAIR}$
- $BASE_{+mBERT}$
- $BASE_{+GBERT}$
- $BASE_{+GELECTRA}$

Since each of them has been trained on 2 datasets, CoNLL-03 and GermEval, there are 12 models in total.

4.1.2 Training and training parameters

Initially, each model setup is trained on Europarl alone using the same parameter configurations as in the first experiment.

Then, each of the existing pre-trained models is finetuned on Europarl, using a learning rate of 0.05 for 50 additional epochs.

Two types of evaluations were performed:

- Cross-corpus evaluation, that is, evaluating the pre-trained models on the test data of Europarl **before** finetuning. I refer to the F-scores as before-finetuning results.
- Evaluation of the models on the test data of Europarl **after** finetuning. I refer to the F-scores as after-finetuning results.

4.1.3 Results

After training and evaluating all setups on the Europarl dataset, the best performing model achieved an F-score of 80.12. This is the baseline against which the after-finetuning results are compared.

Table 4.2 shows the before- and after-finetuning results on the Europarl test data. Every finetuned model outperforms the baseline. Finetuning the models significantly improves their performance on the new domain, compared to their cross-corpus performance. The degree of improvement is indicated by the difference between the F-scores. Large differences show that the models do not generalize well on the new domain, until they are finetuned on it to pick up on new entities.

Table 4.2: Evaluation results on the test set of Europarl before and after finetuning

Model	Source	Target	Before-finetuning	After-finetuning	Difference
BASE	CoNLL2003	Europarl	60.99	82.63	+21,64
BASE$_{+CHAR}$	CoNLL2003	Europarl	68.33	87.18	+18.85
BASE$_{+FLAIR}$	CoNLL2003	Europarl	71.69	91.83	+20.14
BASE$_{+mBERT}$	CoNLL2003	Europarl	75.53	89.16	+13.63
BASE$_{+GBERT}$	CoNLL2003	Europarl	76.24	88.16	+11.92
BASE$_{+GELECTRA}$	CoNLL2003	Europarl	75.62	88.09	+12.47
BASE	GermEval14	Europarl	56.98	84.54	+27.56
BASE$_{+CHAR}$	GermEval14	Europarl	57.97	87.69	+29.72
BASE$_{+FLAIR}$	GermEval14	Europarl	62.89	**93.09**	**+30.2**
BASE$_{+mBERT}$	GermEval14	Europarl	61.66	89.46	+27.8
BASE$_{+GBERT}$	GermEval14	Europarl	62.05	89.37	+27.32
BASE$_{+GELECTRA}$	GermEval14	Europarl	68.95	90.06	+21.11

Observations – The following observations are highlighted:

1. **MISC tags hurt the performance of the models pre-trained on GermEval.** Let's consider the twin models $BASE$, with CoNLL-03 and GermEval as source datasets. Before they get finetuned, they are evaluted on Europarl. $BASE$ pre-trained on CoNLL-03 achieves an F-score of 60.99 and $BASE$ pre-trained on GermEval performs worse, reaching an F-score of 56.98. In order to understand the reason behind the relatively poor performance of the second model, both F-scores are broken down into individual F-scores for each entity type. Figure 4.1 illustrates the difference between scores for each entity type. GermEval's $BASE$ achieves higher scores for PER, ORG and LOC types. It's the MISC tags that substantially decrease the overall score. Charts for other model setups follow the same trend and can be found in the Appendix.

   MISC entities are relatively fewer in number and the hardest to extract, as there is not a proper definition of what constitutes such an entity. Typically, MISC is an entity type which is not a person, location or organization. Events, products, nationalities, works of art can be included in the MISC category. However, every dataset has its own definition of MISC. In CoNLL-03 and Europarl, adjectives related to geopolitics such as "europäischer" or "deutscher" are considered to be MISC entities. In GermEval, such adjectives are not regarded as entities at all, therefore models trained on this dataset will not detect them. As they make up a large portion of MISC entities, the overall performance of models pre-trained on GermEval is greatly affected by them.

2. **Finetuning mitigates the impact of dissimilar entity definitions among datasets.** If models pre-trained on CoNLL-03 perform better on the Europarl before being finetuned, the same does not hold after finetuning. All the models with GermEval as the source

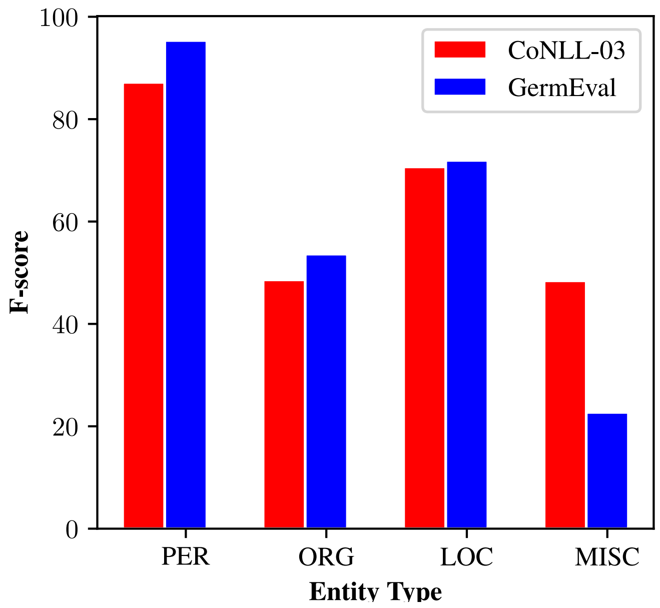


Figure 4.1: Breakdown of the before-finetuning scores for $BASE$. Red and blue represent the scores for the model pre-trained on CoNLL-03 and GermEval, respectively.

dataset outperform their "twins" pre-trained on CoNLL-03. This suggests that finetuning help the models detect new mentions not seen during training on the source dataset. $BASE_{+FLAIR}$, pre-trained on GermEval, performed the best after finetuning (93.09). The same finetuned model, pre-trained on CoNLL-03, had the second best performance.

4.2 Building a multilingual NER model exploiting mBERT

So far, language models have significantly improved the performance of the BiLSTM-CRF architecture on German NER.

Recent research has further expanded their potential, by training language models, which generalise along the language axis [11][35]. Such potential can be used in many sequence labeling applications. In real-life scenarios, we are often faced with texts in different languages. In NER, the standard approach is using a language detection mechanism to choose the appropriate monolingual NER model for the data. This approach requires a series of trained models, embeddings and annotated training data for multiple languages.

Akbik et al. [1] leveraged language modeling with RNNs to model multilingual text at the character level and used it to create a single model for downstream tasks, like PoS-tagging and NER on multilingual text. Instead of Akbik's character-level language model, this work investigates the cross-lingual capabilities of multilingual BERT.

4.2.1 Experimental setup

In order to compare the embeddings from the character-level LM (baseline) and mBERT, I feed them as input to the BiLSTM-CRF module and evaluate it on multiple datasets at once.

- German and English dataset from CoNLL-03 Shared-Task
- Dutch dataset from CoNLL-02 Shared-Task

The datasets are merged, shuffled and used to train the network on a trilingual entity recognition task. Both models are trained for 150 epochs, without changing the hyper-parameters from the first experiment.

4.2.2 Results

The results are presented in Table 4.3 for both approaches. Multilingual BERT embeddings perform better for Dutch, while Akbik's embeddings achieve a higher F-score in English and German. Overall, the NER performance is slightly better, using Akbik's RNN-based language model, with a difference of 0.5 percentage points in F-scores.

Table 4.3: Performance scores for multilingual NER

	Baseline			My approach		
Language	Precision	Recall	F_{score}	Precision	Recall	F_{score}
German	82.93	81.46	**82.19**	84.74	79.55	82.06
English	91.77	92.36	**92.06**	90.05	90.14	90.10
Dutch	89.17	88.76	88.96	89.75	88.66	**89.20**
Overall	88.70	87.56	**88.14**	88.65	86.66	87.64

The character-level language model used by Akbik to generate the word embeddings has been trained on 6 languages: English, German, French, Italian, Dutch and Polish. The training corpus contains data from Wikipedia, parliament speeches, movie subtitles, news commentary and books [1]. On the other side multilingual BERT has been trained on Wikipedia corpora including 104 languages. Both embeddings perform similarly for the task of NER. For German, the baseline outperforms the proposed approach by a small margin (0.13 points). For Dutch, mBERT embeddings seem to be slightly more effective, as they surpass the baseline by 0.24 points. The baseline performs convincingly better for English NER, where the difference is significant, 1.96 points. Multilingual BERT representations have shown to perform cross-lingual generalization well for NER. Therefore mBERT might be a good choice for building NER models that extract entities in not very "well-resourced" languages. However, more experiments need to be conducted to measure how beneficial they are for such scenarios.

Chapter 5

Python-based model integration with DKPro Core

5.1 NER models as part of pipelines

In computing, a pipeline is a chain of data processing elements, arranged so that the output of one element is the input of the next one. NLP pipelines perform sequential processing of speech and text. Each level builds on the output of the previous level, thus breaking down the target task into several smaller tasks. Figure 5.1 shows an example of several components combined together.

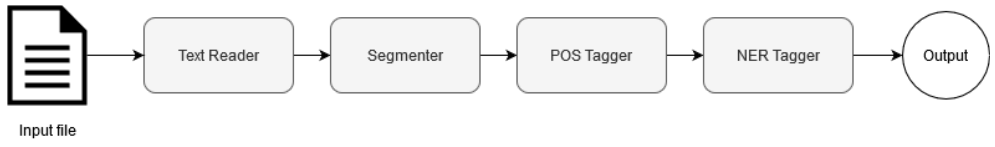


Figure 5.1: An NLP pipeline

For entity extraction, the text is extracted from the input file, broken down into small chunks (i.e. sentences, tokens), analyzed and processed by an NER system, which identifies and labels mentions of entities. They can be useful for other downstream tasks like information extraction, question answering or entity linking. Therefore, NER is often a central component in NLP pipelines.

In science, it's important to share the experimental setups and pipelines with the rest of the research community, to reproduce the results or potentially further improve them. **DKPro Core** has provided a concept for shareable pipelines based on portable components and resources in NLP [13]. The components are distributed via a repository within the Java-ecosystem. In order to make them interoperable, DKPro Core employs Apache UIMA[1].

UIMA ("Unstructured Information Management Architecture") is a platform for unstructured analytics through the reuse of analysis components. Each component implements interfaces defined by the framework and provides self-describing metadata via XML descriptor files. The framework manages the data flow between components.

[1]Ferrucci and Lally, 2004, https://uima.apache.org/

DKPro Core collection includes four third-party NER components from popular NLP tool suites (OpenNLP[2], CoreNLP[3], LingPipe[4], NLP4J[5]). However, they rely on traditional machine learning methods (e.g CoreNLP's CRFClassifier). The recent DL-based approaches have achieved better performance but including them as third-party NLP tools in the collection is not a trivial process. Most state-of-the-art neural approaches are implemented in Python, where the best deep learning libraries (e.g. PyTorch, TensorFlow) are. The authors of DKPro have developed a Python library called **dkpro-cassis**. It is a Python implementation of a special data structure, which represents an object to be enriched with annotations. Because this data structure is defined by UIMA framework, the object enriched with annotations from a DL-based model can be accessed by other UIMA-based components. This eases the integration of Python-based systems in DKPro Core's text analysis workflows.

The following section describes the process of connecting a Python-based NER model with DKpro Core's Java-based components, using dkpro-cassis as an "adapter".

5.2 Workflow

The process is implemented as a Maven project in Eclipse. Maven is a project management tool that is based on POM (Project Object Model). DKPro Core uses the Maven Dependency Plugin to check if all dependencies used directly within the code inside the project are also explicitly declared in the <dependencies> section of POM. The dependencies for the following libraries were declared:

- dkpro-core-io-text-asl
- dkpro-core-opennlp-asl
- dkpro-core-io-xmi-asl

The first processing step is reading the input file. *TextReader* class from 'dkpro-core-io-text-asl' is suitable for reading plain text files. Tokenization is performed by *OpenNlpSegmenter* class from 'dkpro-core-opennlp-asl'. It splits the text into sentences and sentences into tokens, by assigning a start and end offset to each of them. This information is written to an XMI file, using the *XMIWriter* class provided by 'dkpro-core-io-xmi-asl' module. Additionally, a TypeSystem file is generated, whose function becomes clear in the next step.

In a Python script, dkpro-cassis instantiates an object belonging to the before-mentioned data structure. It is referred to as a CAS ("Common Analysis System") object and stores the content from the XMI file generated earlier by the XMI writer, also known as **'Subject of Analysis'** (SofA). The TypeSystem file determines how components store and retrieve their data from the CAS object. Based on the system type, the annotations can be added. Here, the chosen type is NamedEntity, whose annotations consist of a starting position offset, end position offset and a value that represents an entity type (e.g. person, organization etc.)

The NER model is then loaded and it is used to predict the NER tags on the SofA string. For each detected entity, the annotations are created, defined by NamedEntity type. They are added

[2]https://www.tutorialspoint.com/opennlp/opennlp_named_entity_recognition.htm
[3]https://stanfordnlp.github.io/CoreNLP/ner.html
[4]http://www.alias-i.com/lingpipe/demos/tutorial/ne/read-me.html
[5]https://emorynlp.github.io/nlp4j/components/named-entity-recognition.html

to the CAS object, via *add_annotation* method, to be saved later as an XMI file. An XMI reader reads the content of the new file and prints out the detected mentions and their entity type.

Figure 5.2 illustrates the process by showing how components are connected to each other.

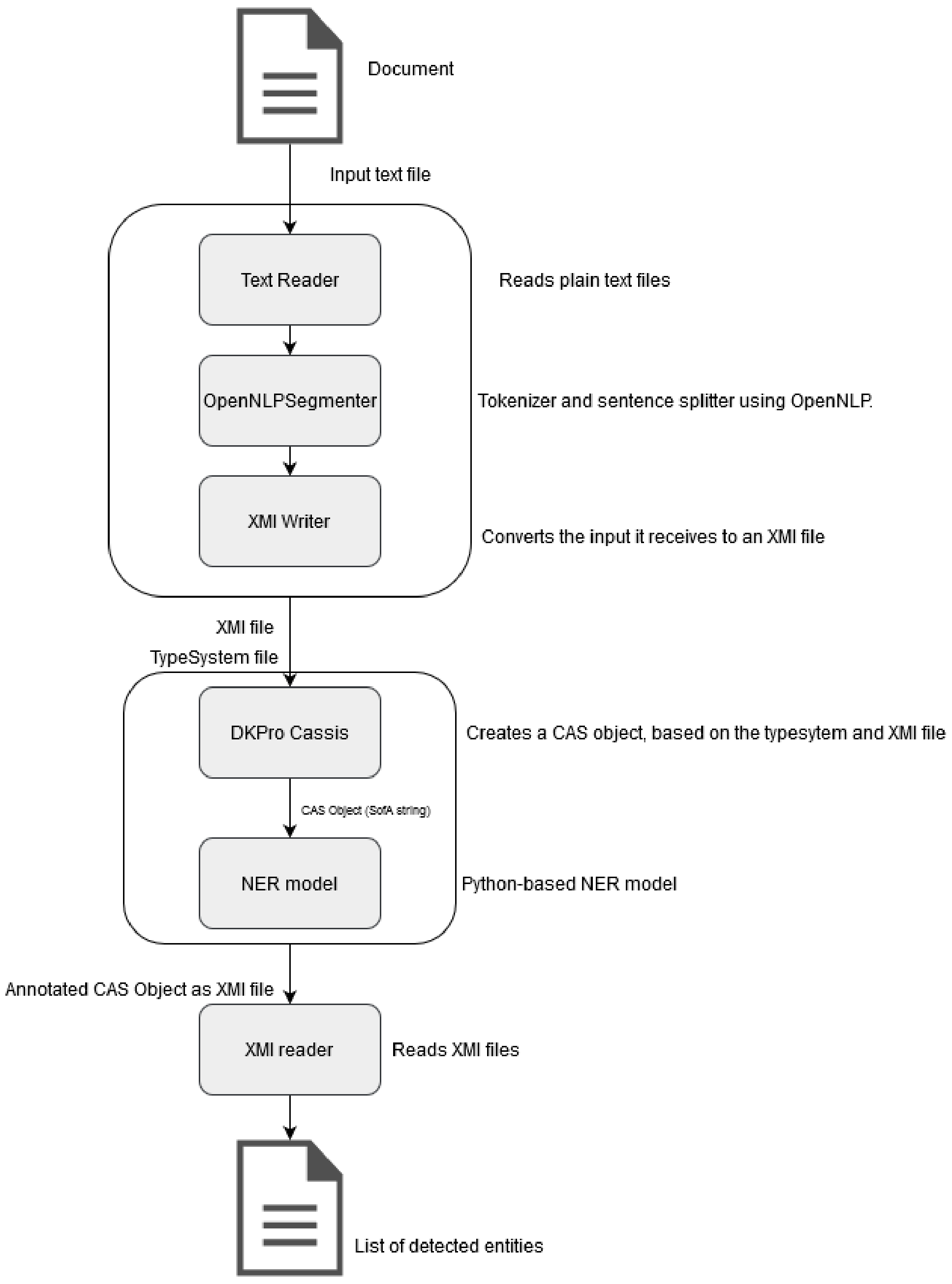

Document
Input text file
Text Reader
Reads plain text files
OpenNLPSegmenter
Tokenizer and sentence splitter using OpenNLP.
XMI Writer
Converts the input it receives to an XMI file
XMI file
TypeSystem file
DKPro Cassis
Creates a CAS object, based on the typesytem and XMI file
CAS Object (SofA string)
NER model
Python-based NER model
Annotated CAS Object as XMI file
XMI reader
Reads XMI files
List of detected entities

Figure 5.2: DKPRo Core Pipeline using a Python-based NER model

5.3 Challenges

Unfortunately, this scheme has some drawbacks, which need to be addressed:

The NER model can be loaded and used for predictions only through Python functions. The ProcessBuilder API creates an operating system process to launch Python and execute the

Python script within a Java process. While this provides high flexibility, regarding different DL-based NER models to be used as annotators, it also introduces a severe overhead, in terms of code and installation procedure. The implementation assumes there is a working Python installation. Furthermore all the relevant Python libraries need to be downloaded beforehand. Additionally, the NER model has to be loaded in the memory, every time the script is executed. This might add a time overhead, in case the NER model is too big. The NER model used in this implementation has a size of approximately 2GB and takes around 5-6 seconds to be loaded and ready to be used.

Chapter 6

Discussion and Conclusion

In this chapter, I summarize the outline of the thesis. Approaches for solving the task of NER have evolved throughout the years, from rule-based approaches to deep neural networks. The recent DL-based approaches have achieved a huge success in many NLP fields. They learn complex features from data via non-linear transformation. They do not rely on handcrafted features, whose design requires domain expertise and engineering skills. The focus is on German NER. Although German is not as widely-spoken as English, it is a well-resourced language, allowing for German NER to receive a fair share of attention. The capitalization of not only proper names, but all nouns and the presence of compound words makes the task difficult in this language particularly.

This thesis quantifies the impact of different types of word embeddings on two domains in German NER. Both non-contextual and contextual embeddings were considered. There were in total six model setups employing the state-of-the-art BiLSTM-CRF architecture with different embedding configurations. They were trained and tested on the two largest available datasets in German, CoNLL-03 and GermEval. All the models performed better on the GermEval dataset. It was shown that contextual embeddings are beneficial for NER models. Transformer-based language models such as GBERT and GELECTRA, pre-trained on German data, achieved the highest scores. To better understand their effectiveness, I conducted a fine-grained analysis to distinguish the effect of these embeddings from the rest. The results show that GBERT/GELECTRA-based embeddings generalize better to entity mentions that span over multiple tokens or mentions with ambiguous entity types (low consistency).

I also improved the performance on Europarl, a low-data domain in German. Finetuning the existing pre-trained models on Europarl led to better results than the models trained on this dataset alone. This shows that 'small-data' scenarios can benefit from using knowledge from a model pre-trained on a bigger dataset. Through finetuning, models learn to identify new examples of entities. This is beneficial, as it allows DL-based models to adjust to recent mentions, using a little amount of training data.

In the third experiment, two single-model approaches to multilingual text data are compared. The difference between the approaches is the type of word embeddings being used. The baseline uses embeddings from an RNN-based LM, pre-trained on six languages, while the proposed approach extracts its embeddings from multilingual BERT, pre-trained on 104 languages. Both models are trained on a mix of data from 3 different languages: German, English and Dutch. The proposed model performed slightly better only on Dutch data, but the overall F-score was higher for the baseline. However, the difference between the scores was tight, showing that multilingual BERT is a good choice for multilingual NER.

The final step is providing a re-usable implementation for one of the pre-trained models from previous experiments. DKPro-Core is an open-source collection of Java-based NLP components, which can be combined into shareable pipelines. Integrating a Python-based NER model in a simple pipeline in this framework requires an "adapter", such that its output can be accessed by the rest of Java-based components. The integration scheme introduces some overhead, in terms of processing time, code and installation procedure.

6.1 Future Work

There is still work to be done. NER is a hot research area, with new developments and improvements happening every year. One interesting aspect to investigate would be whether the temporal factor of the data can be used to obtain better models. During finetuning of pre-trained models on another dataset, temporal information is usually disregarded. However, languages evolve with time, new mentions of entities are added every year and NER models might not generalize well.

Another challenge is integrating a Python-based model in Java-based systems. In this work, the model was accessed through a Python script. However, this is not the most efficient method. An alternative solution is using Deep Java library, a Java framework for deep learning. Because of the limited time and lack of experience with this framework, I was not able to explore this possibility.

Finally, further finetuning of the embeddings on the specific domain can potentially boost the performance of NER models. Additionally, contextualized embeddings from pre-trained XLNet language models have achieved impressive results for NER, while this thesis was being written. Further work could explore recent possibilities for attention/regularization mechanisms to better include context and improve generalization.

Appendices

Appendix A

Appendix

A.1 Bucket-wise analysis

A.2 Breakdown of the before-finetuning results on Europarl.

Tables A.3 and A.4 break down the results in the "Before-finetuning" column of Table 4.2. Red and blue represent the datasets the models have been trained on, repectively CoNLL-03 and GermEval.

A.3 Bucketing Interval Strategy

The bucketing interval with respect to attribute is described here. The range of attribute values is divided into 4 discrete parts. For a given attribute, the number of entities covered by an attribute value varies. For example, for label consistency, there is a large portion of test entities with eCon = 0 and eCon = 1.

- Label consistency (eCon,tCon): The entities in the test set with eCon = 0 and eCon = 1 are put in the first bucket and last bucket, respectively. The rest of the entities are divided equally into 2 buckets. This strategy of eCon is suitable for tCon, too.

- Frequency (eFre,tFre): The entities in the test set with attribute value eFre = 0 go into the first bucket; then, the rest of the entities are equally divided into 3 buckets. This strategy of eFre is suitable for tFre.

Table A.1: Bucket-wise analysis for each model. Numbers from 0 to 3 represent the four buckets, from the smallest to the largest attribute values. The "number/number" pairs in the horizontal axis represent the buckets of a specific attribute on which a model achieved its **worst** and **best** performance, respectively. Green bins represent the difference between the best and worst performance.

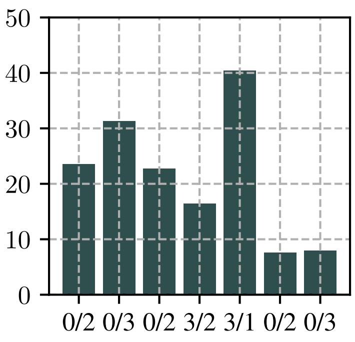
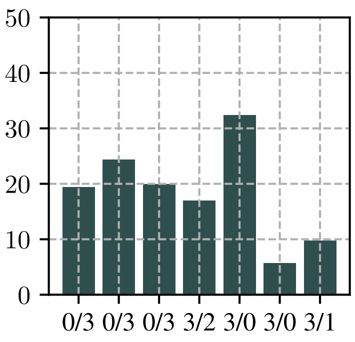
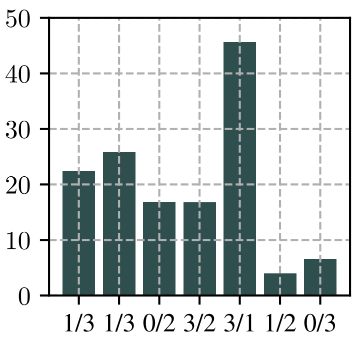
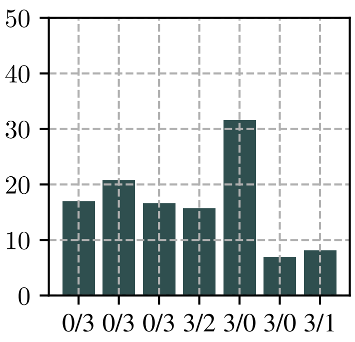
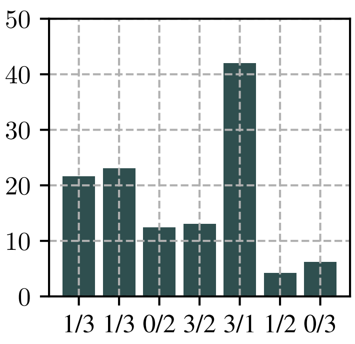
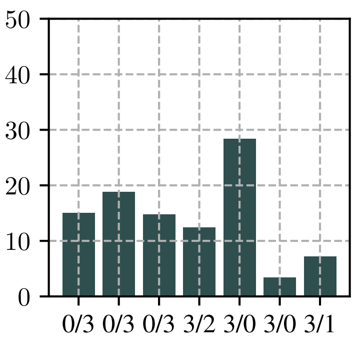
Model	CoNLL-03	GermEval
	eCon tCon eFre tFre eLen sLen eDen	eCon tCon eFre tFre eLen sLen eDen
BASE		
BASE$_{+CHAR}$		
BASE$_{+FLAIR}$		

Table A.2: Continued bucket-wise analysis

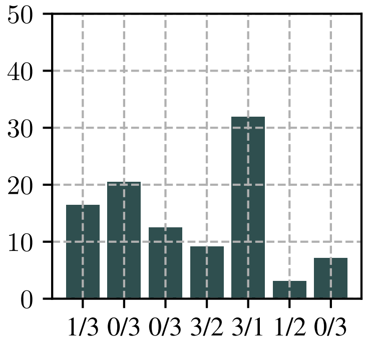
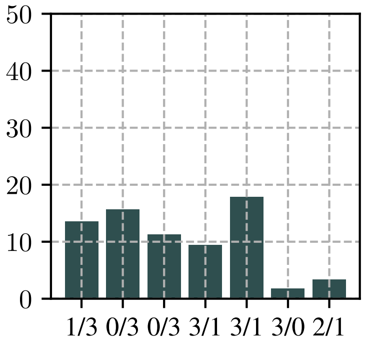
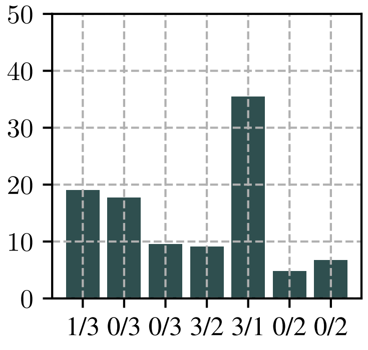
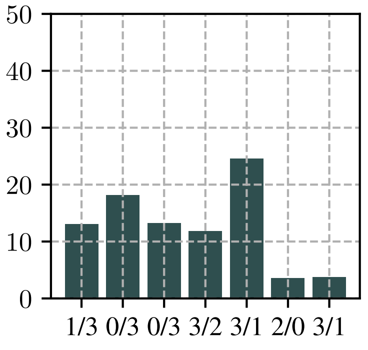
Model	CoNLL-03 (eCon, tCon, eFre, tFre, eLen, sLen, eDen)	GermEval (eCon, tCon, eFre, tFre, eLen, sLen, eDen)
BASE$_{+mBERT}$		
BASE$_{+GBERT}$		

Table A.3: Breakdown of the before-finetuning scores for $BASE_{+CHAR}$, $BASE_{+FLAIR}$ and $BASE_{+mBERT}$. Red and blue represent the scores for the model pre-trained on CoNLL-03 and GermEval, respectively.

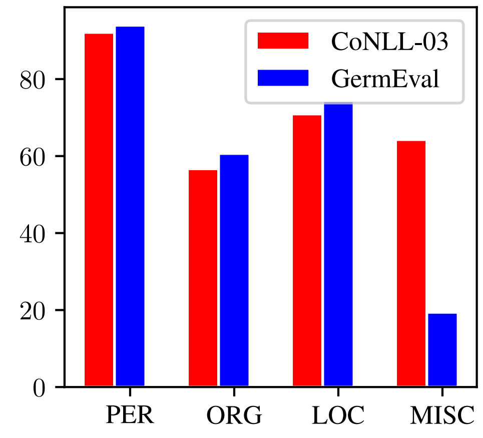
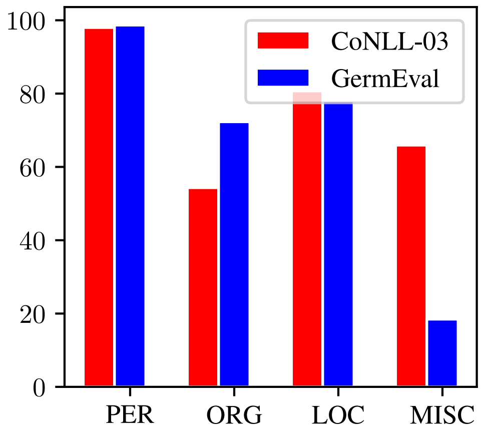
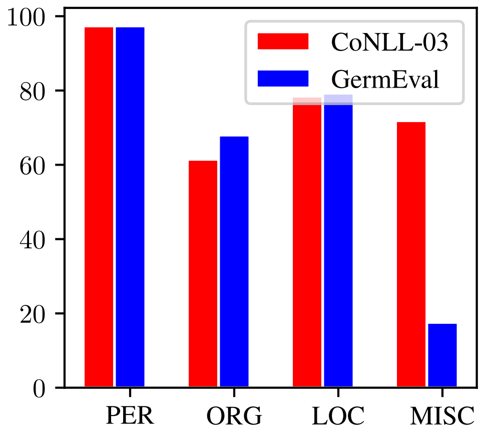
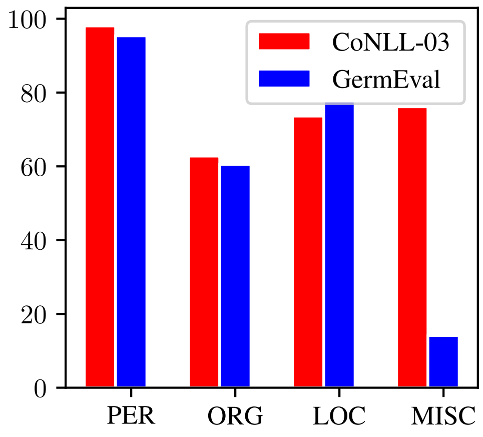
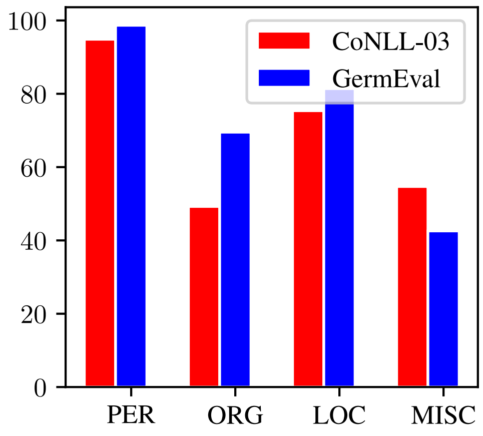
Model	Chart
$BASE_{+CHAR}$	
$BASE_{+FLAIR}$	
$BASE_{+mBERT}$	

Table A.4: Breakdown of the before-finetuning scores for $BASE_{+GBERT}$ and $BASE_{+GELECTRA}$. Red and blue represent the scores for the model pre-trained on CoNLL-03 and GermEval, respectively.

Model	Chart
$BASE_{+GBERT}$	
$BASE_{+GELECTRA}$	

- Sentence length (sLen) and entity density (eDen): the test entities are divided equally into 4 buckets.

- Entity length (eLen): Generally, entities do not span over more than 4-5 words, however there are rare cases when they do. Therefore, the entities in the test set with lengths of 1, 2, 3, and longer than 4 are separated into four buckets, respectively.

The bucket values for each attribute are given in the tables below:

Table A.5: Boundary values for GermEval

Attribute	Bucket 0	Bucket 1	Bucket 2	Bucket 3
eCon	[0,]	]0,0.5]	]0.5,0.999]	[1,]
tCon	[0,]	]0,0.5]	]0.5,0.999]	[1,]
eFre	[0,]	]0,0.008]	]0.008,0.0056]	]0.056, 0.973]
tFre	[0,]	]0, 0.002]	]0.002, 0.013]	]0.013, 1]
eLen	[1,]	[2,]	[3,]	[4,20]
sLen	[1,16]	[17, 23]	[24, 31]	[32, 48]
eDen	[0.0217, 0.0952]	]0.0952, 0.1515]	]0.1515, 0.2381]	]0.2381, 0.8889]

Table A.6: Boundary values for CoNLL

Attribute	Bucket 0	Bucket 1	Bucket 2	Bucket 3
eCon	[0,]	]0,0.5]	]0.5,0.999]	[1,]
tCon	[0,]	]0,0.5]	].5,0.999]	[1,]
eFre	[0,]	]0,0.013]	]0.013, 0.057]	]0.057, 0.786]
tFre	[0,]	]0, 0.002]	]0.002, 0.016]	]0.016, 1]
eLen	[1,]	[2,]	[3,]	[4,9]
sLen	[1,15]	[16, 24]	[25, 33]	[34, 107]
eDen	[0.0161, 0.1111]	]0.1111, 0.1714]	]0.1714, 0.2727]	]0.2727, 1]

The code and the best performing models for German NER can be accessed through this **Google Drive link**.

Bibliography

[1] A. Akbik, Tanja Bergmann, and Roland Vollgraf. Multilingual sequence labeling with one model. 2019.

[2] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

[3] Y. Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5:157–66, 02 1994.

[4] Darina Benikova, Chris Biemann, and Marc Reznicek. NoSta-D named entity annotation for German: Guidelines and dataset. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2524–2531, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA).

[5] Darina Benikova, Seid Muhie Yimam, Prabhakaran Santhanam, and Chris Biemann. Germaner: Free open german named entity recognition tool. In Bernhard Fisseni, Bernhard Schröder, and Torsten Zesch, editors, *Proceedings of the International Conference of the German Society for Computational Linguistics and Language Technology, GSCL 2015, University of Duisburg-Essen, Germany, 30th September - 2nd October 2015*, pages 31–38. GSCL e.V., 2015.

[6] Daniel M. Bikel, Richard M. Schwartz, and Ralph M. Weischedel. An algorithm that learns what's in a name. *Mach. Learn.*, 34(1-3):211–231, 1999.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguistics*, 5:135–146, 2017.

[8] Branden Chan, Stefan Schweter, and Timo Möller. German's next language model. In Donia Scott, Núria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 6788–6796. International Committee on Computational Linguistics, 2020.

[9] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[10] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. *Proceedings of the 1999 Joint SIGDAT Conference on EMNLP and VLC*, 12 2002.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[12] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 334–343. The Association for Computer Linguistics, 2015.

[13] Richard Eckart de Castilho and Iryna Gurevych. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University.

[14] Oren Etzioni, Michael J. Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artif. Intell.*, 165(1):91–134, 2005.

[15] Manaal Faruqui and Sebastian Padó. Training and evaluating a german named entity recognizer with semantic generalization. In *Proceedings of KONVENS 2010*, Saarbrücken, Germany, 2010.

[16] Jinlan Fu, Pengfei Liu, and Graham Neubig. Interpretable multi-dataset evaluation for named entity recognition. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6058–6069. Association for Computational Linguistics, 2020.

[17] Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. Multilingual language processing from bytes. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1296–1306. The Association for Computational Linguistics, 2016.

[18] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005. IJCNN 2005.

[19] Ralph Grishman and Beth Sundheim. Message Understanding Conference- 6: A brief history. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996.

[20] Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 267–274, New York, NY, USA, 2009. Association for Computing Machinery.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[22] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991, 2015.

[23] Jun'ichi Kazama and Kentaro Torisawa. Exploiting Wikipedia as external knowledge for named entity recognition. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 698–707, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[24] Yoon Kim, Yacine Jernite, David A. Sontag, and Alexander M. Rush. Character-aware neural language models. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2741–2749. AAAI Press, 2016.

[25] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 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, June 2016. Association for Computational Linguistics.

[26] Wenhui Liao and Sriharsha Veeramachaneni. A simple semi-supervised algorithm for named entity recognition. In *Proceedings of the NAACL HLT 2009 Workshop on Semi-supervised Learning for Natural Language Processing*, pages 58–65, Boulder, Colorado, June 2009. Association for Computational Linguistics.

[27] Wang Ling, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. Finding function in form: Compositional character models for open vocabulary word representation. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1520–1530. The Association for Computational Linguistics, 2015.

[28] Xuezhe Ma and Eduard H. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.

[29] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 188–191, 2003.

[30] Paul McNamee and James Mayfield. Entity extraction without language-specific resources. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*, 2002.

[31] Andrei Mikheev. A knowledge-free method for capitalized word disambiguation. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 159–166, College Park, Maryland, USA, June 1999. Association for Computational Linguistics.

[32] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[33] David Nadeau, Peter D. Turney, and Stan Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In Luc Lamontagne and Mario Marchand, editors, *Advances in Artificial Intelligence*, pages 266–277, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[34] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[35] Telmo Pires, Eva Schlinger, and Dan Garrette. How multilingual is multilingual bert? In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4996–5001. Association for Computational Linguistics, 2019.

[36] Yael Ravin and Nina Wacholder. Extracting names from natural-language text. 11 1998.

[37] Hadas Raviv, Oren Kurland, and David Carmel. Document retrieval using entity-based language models. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 65–74, New York, NY, USA, 2016. Association for Computing Machinery.

[38] Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[39] Matthias Richter, Uwe Quasthoff, Erla Hallsteinsdóttir, and Chris Biemann. C.: Exploiting the leipzig corpora collection. 01 2006.

[40] Martin Riedl and Sebastian Padó. A named entity recognition shootout for German. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 120–125, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[41] Stefan Schweter and Johannes Baiter. Towards robust named entity recognition for historic german. In Isabelle Augenstein, Spandana Gella, Sebastian Ruder, Katharina Kann, Burcu Can, Johannes Welbl, Alexis Conneau, Xiang Ren, and Marek Rei, editors, *Proceedings of the 4th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2019, Florence, Italy, August 2, 2019*, pages 96–103. Association for Computational Linguistics, 2019.

[42] Satoshi Sekine and Chikashi Nobata. Definition, dictionaries and tagger for extended named entity hierarchy. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal, May 2004. European Language Resources Association (ELRA).

[43] Satoshi Sekine and Elisabete Ranchhod. Named entities: recognition, classification and use., Jul 2009.

[44] Burr Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP)*, pages 107–110, Geneva, Switzerland, August 28th and 29th 2004. COLING.

[45] Erik F. Tjong Kim Sang and Fien De Meulder. 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*, pages 142–147, 2003.

[46] Antonio Toral and Rafael Muñoz. A proposal to automatically build and maintain gazetteers for named entity recognition by using Wikipedia. In *Proceedings of the Workshop on NEW TEXT Wikis and blogs and other dynamic text sources*, 2006.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[48] Vikas Yadav and Steven Bethard. A survey on recent advances in named entity recognition from deep learning models. In Emily M. Bender, Leon Derczynski, and Pierre Isabelle,

editors, *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 2145–2158. Association for Computational Linguistics, 2018.

[49] Vikas Yadav, Rebecca Sharp, and Steven Bethard. Deep affix features improve neural named entity recognizers. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 167–172, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[50] Hirotoshi Yamada, T. Kudo, and Y. Matsumoto. Japanese named entity extraction using support vector machine. 01 2001.

[51] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764, 2019.

[52] Shaodian Zhang and Noémie Elhadad. Unsupervised biomedical named entity recognition: Experiments with clinical and biological texts. *Journal of Biomedical Informatics*, 46(6):1088–1098, 2013. Special Section: Social Media Environments.

[53] G. Zhou and J. Su. Named entity recognition using an hmm-based chunk tagger. In *Proceedings of the 40th Annual Annual Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia,PA, 2002.

[54] Jianhan Zhu, Victoria S. Uren, and Enrico Motta. Espotter: Adaptive named entity recognition for web browsing. In Klaus-Dieter Althoff, Andreas Dengel, Ralph Bergmann, Markus Nick, and Thomas Roth-Berghofer, editors, *Professional Knowledge Management, Third Biennial Conference, WM 2005, Kaiserslautern, Germany, April 10-13, 2005, Revised Selected Papers*, volume 3782 of *Lecture Notes in Computer Science*, pages 518–529. Springer, 2005.