University of Zurich[UZH]

Master's thesis
for the degree of
Master of Arts
presented to the Faculty of Arts and Social Sciences
of the University of Zurich

Toulouse and Cahors are French Cities, but T«i*louse and Caa.Qrs as well

A Neural Approach for Detecting Named Entities in Digitized Historical Newspapers

Author: Stefan Bircher
Student ID: 08-705-741

Examiner: Dr. Prof. Martin Volk

Supervisor: Dr. Simon Clematide

Institute of Computational Linguistics

Submission date: 10.07.2019

Abstract

Named entity recognition (NER) in digitized historical texts faces the challenge of dealing with words and multi-word expressions which are characterized by spelling variation originating from missing standard orthographies and/or errors introduced by optical character recognition preprocessing. In this thesis, NER is considered as a sequence labelling problem which is approached by means of neural sequence classifiers and the focused data corresponds to a collection of French newspaper issues from the late 19th century. The employed sequence models rely on particular word representations which are generated by means of neural character-level language modes. Such language models, trained to predict the next character in a sequence based on the preceding characters, facilitate the creation of word embeddings which model words as strings of contextualized characters. Since these contextualized string embeddings are robust in face of misspelled words, they are highly suitable for NER in digitized historical data. Compared to classical feature-based sequence classifiers, neural models which rely on contextualized string embeddings achieve performance gains of up to 12 points F_1 on the used historical newspaper corpus.

Zusammenfassung

Eigennamenerkennung in digitalisierten historischen Texten wird erschwert durch den Umstand, dass für Wörter und Mehrwortausdrücke oftmals verschiedene Schreibweisen vorhanden sind. Dieses Phänomen ist auf das Fehlen einheitlicher Rechtschreiberegeln und/oder fehlerhafte Zeichenerkennung bei der Vorverarbeitung der Daten zurückzuführen. In dieser Arbeit wird Eigennamenerkennung als ein Sequenzklassifikationsproblem betrachtet, welches mittels neuronalen Sequenzklassifikatoren in Angriff genommen wird. Das benutzte Datenmaterial entspricht einer Sammlung von französischen Zeitungsausgaben vom Ende des 19. Jahrhunderts. Die verwendeten Sequenzmodelle beruhen auf speziellen Wortrepresentationen, welche anhand neuronaler Sprachmodelle generiert werden, die darauf trainiert sind, das nächste Zeichen in einer Sequenz beruhend auf den vorangehenden Zeichen vorauszusagen. Diese Sprachmodelle ermöglichen die Erzeugung von *Contextualized String Embeddings*, welche Wörter als Folgen von kontextualisierten Zeichen darstellen. Da diese *Embeddings* robust gegenüber falsch geschriebenen Wörtern sind, erweisen sie sich als höchst geeignet für die Eigennamenerkennung in digitalisierten historischen Daten. Im Vergleich zu merkmalsbasierten Sequenzklassifikatoren erreichen auf *Contextualized String Embeddings* basierende Modelle Performancegewinne von bis zu 12 F_1-Punkten auf dem verwendeten historischen Korpus.

Acknowledgement

In the first place, I would like to thank Dr. Simon Clematide for his guidance and patience and, most importantly, for offering me the opportunity to work on this highly interesting research project.
My sincerest gratitude goes to Oma, Opa and Juliana for the countless hours of taking care of Laura and Luis, and for everything else they have done for me so far.

Contents

List of Figures

List of Tables

List of Acronyms

CER	Character Error Rate
CRF	Conditional Random Filed
FFNN	Feed-Forward Neural Network
LM	Language Model
LSTM	Long-Short Term Memory
NE	Named Entity
NER	Named Entity Recognition
NLP	Natural Language Processing
NN	Neural Network
OCR	Optical Character Recognition
OLR	Optical Layout Recognition
POS	Part-Of-Speech
QOP	Quaero Old Press
RNN	Recurrent Neural Network
UTF-8	Unicode Transformation Format (8-bit)
WER	Word Error Rate
XML	eXtensible Markup Language

1 Introduction

1.1 Motivation

In natural language processing (NLP), formal models and algorithms are devised and implemented in order to facilitate computers to exploit, process and even produce spoken or written language data. Toward this main objectives, the NLP pipeline assembles a series of tasks which build, to a greater or lesser extent, on the outputs of previous ones. A such task is represented by named entity recognition (NER), the identification and categorization of named entities, e.g., person names or mentions of locations, in a text. With respect to the dependence on other NLP tasks, NER may be implemented as presupposing text annotated with Part-of-Speech tags and can provide, for its part, the input for subsequent tasks like relation extraction, sentiment analysis or question answering.

Traditional approaches to tackle NER tasks involve rule-based methods, or, if considered as a sequence labelling problem, statistical models to detect and label the focused elements. Both of them either require sophisticated patterns that match the character and word-level properties of named entities or rely on extensive feature engineering which allows for useful word representations. Additively, external resources such as lexicons or gazetteers may be consulted to enhance the retrieval of entities. In the end, NER systems relying on such methods often restrict the scope of application to a specific language, domain and genre for which the rules and features are construed.

While many of these implementations are designed to deal with contemporary input data, i.e., data attributable to late 20th and 21th centuries' standard languages, the processing of historical documents implies dealing with certain peculiarities which are not adequately covered by systems designed for modern texts. Older language states are often characterized by a lack of standardization, resulting in changing orthographic an syntactic conventions. Moreover, words can't be considered as morphologically and semantically static elements since their spellings and meanings change over time. Accordingly, a word may adopt different formings, which can even vary in one and the same text, and possibly occurs in diverging contexts across

time.
Variance in terms of word spellings can appear further intensified by reason of another aspect of historical documents, or, rather, their digital versions used by NLP applications. Whereas contemporary language data is usually provided in already digitized from, as it's the case for corpora compiled from Internet resources, numerous centuries-old documents are preserved physically in library archives. At most, they are available in form of digital images, which have to be previously processed by optical layout and optical character recognition systems in order to generate machine readable input data. This pre-processing steps tend to introduce recognition errors and noise, especially if the depicted documents are in bad condition or the texts are written in black letter typesets.
With regard to more recent approaches, the potential incompatibility between features engineered for contemporary data and historical texts can be considered superfluous. In contrast to feature-based models, systems based on neural networks show a general tendency to completely omit linguistic knowledge as well as other external resources, being word embeddings the only utilities admitted in most of the cases. By this means, neural models allow for automatically learning word representations from the target data itself, either in form of an upstream task or as part of an end-to-end system. Effective methods to profitably incorporate morphological and syntactical information encode words or characters as the aggregate of contexts they occur in. In this thesis, an implementation of word representations is described which is robust in the face of spelling variation and noise introduced by optical character recognition.

1.2 Task and Research Questions

In the present work, NER in historical digitized texts is approached as a sequence labelling problem. This task of assigning a label to each element in a sequence, e.g., to the words in a sentence, is tackled by means of neural sequence models. In this connection, neural networks or, more precisely, recurrent neural networks prove especially useful since they facilitate modelling long-distance dependencies between the words in a sentence. These networks rely on word embeddings, which represent words as vectors of real numbers. The neural approach focused in this thesis uses a particular embedding type based on neural character-level language models. Such language model, trained to predict the next character in a sequence given the preceding characters, holds a hidden state for each character in a sequence. A hidden state encodes a character's entire history, i.e., the context from the beginning or the

end of the sequence up the character itself. Thus, words represented as their characters' hidden states are basically modelled as strings of contextualized characters and are able to "both better handle rare and misspelled words as well as model subword structures such as prefixes and endings" (Akbik et al., 2018, p. 1939). Actually, these contextualized string embeddings seem almost predestined for representing words in historical texts and are accordingly employed for this work.
The NER task as a whole is performed and evaluated on named entity annotated data which was prepared in the context of a French research program instituted to promote the development of multimedia and multilingual indexing tools. The chosen corpus can be considered historical as it consists of a collection of newspaper documents from the late 19th century. Due to the already advanced language standardization in the respective time period, it is not assumed that historical spelling variation represents a crucial factor. However, irregularities introduced by means of optical layout and character recognition are supposed to have a noticeable impact on the quality of the data.
Given the indicated word representations in form of contextual string embeddings and the digitized historical French newspaper corpus, the research questions are formulated as follows:

1. To what extent do contextual string embeddings from character-level language models trained on noisy in-domain data favour NER in OCRed historical texts? Is there a noticeable difference if the underlying language models are trained on clean out-of-domain data? In how far does mixed-domain language model training data affect the neural sequence model's performance?

2. How well performs a neural sequence model which relies on contextual string embeddings compared to a feature-based classifier? Are there named entity categories which can be better handled by one or other of the models?

3. Are there noticeable performance losses if the neural sequence model which relies on contextual string embeddings is applied to data different from the one it was trained on? How compares the neural classifier to a feature-based model with respect to this shift?

1.3 Thesis Structure

In Chapter 2, it is given an extended overview of the characteristics of digitized historical texts and the challenges they pose for NER and other NLP related tasks. In addition, the actual NER task is examined in greater detail. To this, some first ex-

amples from the used French newspaper corpus are adduced. Chapter 3 introduces the necessary theoretical background with respect to NER devised as a sequence labelling task. Moreover, it is provided the fundamental understanding of neural networks in general and recurrent neural networks in particular. Chapter 4 indicates previous work with regard to, on the one hand, NER in digitized historical data and, on the other hand, sequence labelling based on neural models. In Chapter 5, the utilized corpus is presented and the essential data preprocessing steps are delineated. Furthermore, it are described the neural sequence and language models employed in the context of this thesis as well as the central computational tools to which was resorted in order to implement them. Chapter 6 presents the outcomes of the experiments conducted to determine the neural models which perform best on the OCRed data. In Chapter 7, the results in the form of a performance comparison between the neural models which rely on contextual string embeddings an feature-based classifiers are discussed. Moreover, the models' performances for misspelled entities are examined. Eventually, Chapter 8 concludes this thesis with some closing words.

2 Approaching NER in Historical Data

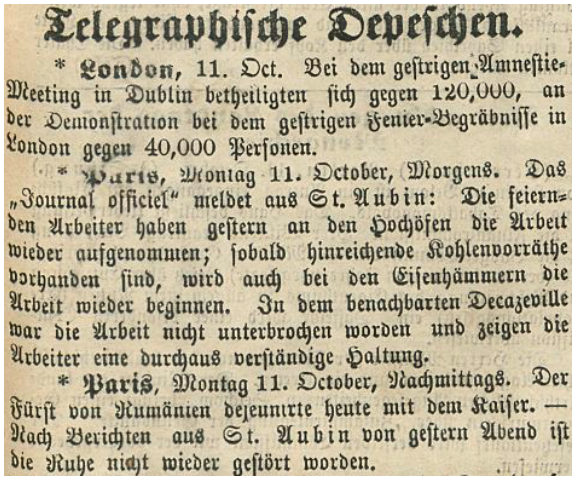
In Chapter 1, NER has been introduced as a task aimed at detecting and labelling named entity occurrences in text data. Likewise, it has been indicated that the application of NER, and generally any other NLP task, to digitized historical documents entails some fundamental challenges. This chapter amplifies the understanding of digitally processed and preserved historical data and its characteristics affecting NLP tasks. Subsequently, the initial definition of NER is elaborated more comprehensively and the respective implementation as well as its use case are commented in greater detail.

2.1 Digitized Historical Texts

In general, historical texts can be determined as documents which are distinguished as being written in an out-of-use language that is attributable to a past epoch. Following Piotrowski (2012, p. 2f.), such historical languages oppose modern languages in the sense that they neither have standard variants, i.e., variants serving as common literary languages or for information exchange on national level, nor do they obey consistent orthographies. This makes it difficult, if not impossible, for present-day readers to decode the texts in question. However, the categorization of a specific document's language as a historical one is not always clear from the outset. Likewise, the identification and delimitation of such antiquated languages may represent challenging tasks in many cases. Whereas, for instance, Old German or Old French, to be found in medieval chronicles or heroic epics, could unequivocally be classified as historical, this is not the case for languages encountered in German or French newspaper issues from the turn of the 20th century. Hence, understanding writings originating from the Middle Ages represents a rather difficult exercise, even for skilled readers. By contrast, the articles in a late 19th century journal, although written more than a hundred years ago, shouldn't pose a problem for most contemporary native speakers.

Recurring to a simple example from the *Breve diccionario etimológico de la lengua castellana*, for the Spanish word *palabra* ('word'), which has its roots in the Latin

expression *parabŏla* ('comparison', 'simile') and is first attested by 1140, an archaic form *parabla* can be found in documents from about 1250 and the spelling variant *palavra* existed until the beginning of the 17th century. In addition to the synchronic and diachronic alteration concerning its form, the term passed through a semantic change from the original Latin meaning to 'phrase' between the 12th and 14th centuries and finally to 'vocable' and 'word' in modern Spanish. Oftentimes, the consolidation of a language's vocabulary, i.e., the words' spellings and their meanings, as well as its syntax was initialized by the first publication of a grammar, such as the *Gramática castellana* in 1492 or the *Tretté de la Grammaire française* in 1550. Alongside the efforts of official institutions, national publishing houses, coming into existence in the aftermath of the invention of the printing press in the 15th century, familiarized a broader audience with increasingly consistent spelling norms. Thus, the rather smooth legibility of 19th century newspaper articles is given by the fact that the standardization processes, undergone by most European languages with a standard variety, are at an already advanced stage by 1900.



Telegraphiſche Depeſchen.

* London, 11. Oct. Bei dem geſtrigen Amneſtie-Meeting in Dublin betheiligten ſich gegen 120,000, an der Demonſtration bei dem geſtrigen Fenier-Begräbniſſe in London gegen 40,000 Perſonen.

* Paris, Montag 11. October, Morgens. Das „Journal officiel" meldet aus St. Aubin: Die feiernden Arbeiter haben geſtern an den Hochöfen die Arbeit wieder aufgenommen; ſobald hinreichende Kohlenvorräthe vorhanden ſind, wird auch bei den Eiſenhämmern die Arbeit wieder beginnen. In dem benachbarten Decazeville war die Arbeit nicht unterbrochen worden und zeigen die Arbeiter eine durchaus verſtändige Haltung.

* Paris, Montag 11. October, Nachmittags. Der Fürſt von Rumänien dejeunirte heute mit dem Kaiſer. — Nach Berichten aus St. Aubin von geſtern Abend iſt die Ruhe nicht wieder geſtört worden.

Figure 1: Extract of an issue of the *Hannoverscher Courier* from 12 October 1869, which is provided by the *ZEFYS Zeitungsinformationssystem*[1].

A 19th century newspaper article, notwithstanding its proximity to contemporary writings, may still show sporadic discrepancies in comparison to a today's counterpart. Regarding the text fragment in Figure 1, which dates from 12 October 1869, words containing a long *s* (*ſ*) as well as instances such as 'Kohlenvorräthe' ('coal reserves') and 'dejeunirte' ('(he) had breakfast') can be identified as adhering to outdated spelling variants. The relevant rules which determine the modern spellings 'Kohlenvorräte' and 'dejeunierte' were only established in the context of the German Orthographic Conference of 1901. Moreover, the term *dejeunieren* has generally fallen out of use and is hardly expected to appear in any present-day document.

From the NLP perspective, absence of standard variants and lack of uniform orthographies, still observable in 19th century documents, are not the only characteristics which make historical languages differ from contemporary ones. On the one hand, large-scale machine-readable document collections, such as *Google Books*[2], have only recently been created and are not always freely available. On the other hand, there are few to no instruments attuned to automatically process historical data. The bulk of ready to use NLP tools, implementing either rule-based or statistical techniques, is developed for or trained on modern standard language data and usually builds on the premise that each lexical item has its own distinctive form. Thus, unless adapted in order to cope with, for instance, spelling variation and outdated vocabularies, these resources and procedures are expected to produce noticeably worse results when applied to historical texts. Moreover, the lack of electronically available and preferably annotated corpora represents an impeding factor for the elaboration of tools tailored to historical documents since effective approaches to tackle NLP tasks by means of, for example, machine learning techniques require huge amounts of labelled data to learn from. However, the shortage in machine-readable historical material appears remediated, at least partly, in the light of growing digital text collections such as the *HathiTrust Digital Library*[3] or the *Europeana Collections*[4]. These electronic libraries incorporate the outcomes from recent digitization endeavours expedited by private or public organizations and tend to cover a variety of domains and genres as well as different historical languages.
As, by nature, historical documents "were not born digital" (Piotrowski, 2012, p. 4), they have to pass through an essential work flow in order to being converted into digital text. This procedure comprises the initial digitization of a document's original print version, which is usually realized by means of an image scanner, and the subsequent processing of the generated image with Optical Layout Recognition (OLR) and Optical Character Recognition (OCR). Whereas OLR aims at determining and keeping apart different document sections, such as headers, footers, captions and column-wise organized contents, the actual text or, more precisely, the contained characters and words are identified by OCR. For both processes, which are often combined in a single system, the condition of the original document as well as the quality of the corresponding digital image constitute decisive criteria for the recognition reliability. Further important factors, particularly in the case of OCR, are represented by the text's style of lettering and, if the system relies on supplementary resources when selecting from among multiple potential recognition candidates, the language in which it's composed. Though, while some systems such

[2]https://books.google.com/

[3]https://www.hathitrust.org/

[4]https://www.europeana.eu/portal/

as the *ABBYY FineReader* are able to handle black letter fonts, it would pose a rather insurmountable challenge to create, for example, a subsidiary dictionary that satisfactorily covers historical spelling variants (Piotrowski, 2012, p. 31–33).
Given, on the one hand, the original documents' state of preservation, which is frequently marked by physical damage, contamination as well as faded writing, and, on the other hand, spelling variation and outdated vocabularies, OCR in historical data often produces defective output. Hence, depending on these factors, the generated electronic texts exhibit a varying portion of noisy and miss-recognized characters. Instances of recognition errors attributable to the original printing's condition are illustrated by means of the four images sections in Figure 2 and the corresponding OCR output shown in the examples (2.1) to (2.4).

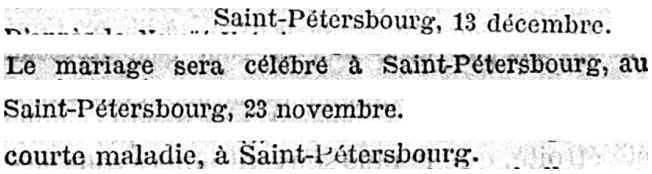


Figure 2: Occurrences of the expression *Saint-Pétersbourg* in four non-contiguous image sections which were used in the context of the OCR preprocessing of the French *Quaero Old Press* corpus.

(2.1) tv •= i at *Saint:Pétcrsbourg*, 13 décembre.

(2.2) Le mariage sera célébré à *SainMPétersbourg*, au

(2.3) *Saint-Pétersbourg1*, 23 novembre.

(2.4) courte maladie, à *Saint-lJôtcrsbourg*.

The above instance of erroneous OCR correspond to four discontiguous mentions of the Russian city of Saint Petersburg in the French *Quaero Old Press* (QOP) corpus[5]. Without going into the causes of the recognition errors, for the OCRed late 19th century documents of the QOP corpus can be stated that 'Saint:Pétcrsbourg', 'SainM-Pétersbourg', 'Saint-Pétersbourg1' and 'Saint-lJôtcrsbourg' represent four variants of the predominantly occurring spelling 'Saint-Pétersbour'. Accordingly, spelling variation introduced by faulty OCR adds up to the historically conditioned one, which complicates subsequent processing of digitized historical data even further.
While typewriting dozens or even hundreds of printed documents generally does not represent a feasible alternative when digitizing larger amounts of text, manual correction of OCR output, e.g., with the help of crowdsourcing methods (see

[5]http://catalog.elra.info/en-us/repository/browse/ELRA-W0073/

Holley, 2010), is possibly worth considering. Likewise, automatic correction of miss-recognized words could be an option. Here, machine learning techniques as well as rule-based approaches, designed to exploit the knowledge of the OCR system's frequently committed mistakes or character swaps, are potential candidates. However, the recourse to such downstream manual or automatic correction processes implies additional resource costs, which maybe are not available. Furthermore, historical spelling variation, unless eliminated by means of an additive normalization procedure (see Bollmann, 2019), still remains present and potentially lowers the effectiveness of non-adapted customary NLP tools.
Ultimately, approaches which take into account that a word can adopt many different forms, including either accidentally created or historically conditioned ones, constitute a valuable alternative. As indicated in Chapter 1, the focus of this work lies on such a method that is robust to spelling variation and, therefore, OCR correction or spelling normalization in digitized historical texts are not followed up any further. Likewise, in contrast to the hitherto adduced multilingual exemplifications and visualizations, subsequent instancing recurs basically to French language examples. More precisely, since the material basis of this work corresponds to the data from the QOP corpus, illustrations are taken, unless otherwise noted, from the 19th century French newspaper collection.

2.2 Named Entity Recognition

Named entity recognition comes up to the NLP task of recognizing those elements in a document that point to a unique entity, such as an individual, a concrete place or a specific corporation, and classifying them according to a predefined set of categories. Thus, to the process as a whole can be ascribed, on the one hand, the task of detecting relevant words or multi-word terms in a text and delimiting them from non-relevant ones. In (2.5), for example, the entire expression *boulevard Saint-Germain, 262* refers to a specific place or, more precisely, a concrete address and should be recognized as forming one single entity mention.

(2.5) On la trouve à l'œuvre des Cercles, *boulevard Saint-Germain, 262*.

On the other hand, the correct labels have to be assigned to the identified elements, also in the case of ambiguity arising from entities belonging to different types but named with one and the same expression. For instance, the term *Saint-Germain* in (2.6) is used to denote the starting point of a diligence route, located in a Parisian district, and is accordingly classified as a location. In (2.7), on the contrary, labelling

the identical expression as a particular place would turn out rather inappropriate. Here, *Saint-Germain* corresponds to a family name, which refers to a French stage actor who forms part of the cast of the theater play *Un Prix Montyon*, and should be correctly categorized as belonging to the *person* class.

(2.6) Justement ce même jour, 1er juillet, une chaise de poste, lancée au grand trot de quatre vigoureux percherons, roulait sur la route qui mène de *Saint-Germain* à Versailles.

(2.7) *Saint-Germain* a fait du professeur de morale une caricature très étudiée et très amusanté.

The basic understanding of a named entity (NE), approaching the definition of a proper noun, can appear broadened to "any type of mono- or multi-word expression which designates an object or concept of the real world that belongs to a class of potential interest for a given application" (Galibert et al., 2010, p. 3453). Hence, depending on the application area, NER may also consider words that refer to date and time indications, any kind of amounts, e.g., of goods, taxes or costs, or even chemical agents and biomedical substances when dealing with scientific domains (Jurafsky and Martin, 2018, p. 327–329).
Likewise, a more comprehensive set of labels can be established by subdividing the traditional entity types *person*, *location* and *organization* into more fine-grained categories. If required, classes such as *town*, *region* and *nation* may be employed instead of the general *location* type. Eventually, the labels to be used are defined in often project-specific annotation guidelines, which are usually designed to support human annotators in, for example, the preparation of a gold standard, i.e., a manually annotated data set intended to train a NER system or to evaluate its performance.
In addition to the detailed set of categories, annotation guidelines commonly contain further information about what actually belongs to an entity and what should be excluded. In order to consistently implement the delimitation task indicated above, it has to be specified, for instance, whether the post modifying clause 'commandant supérieur du cercle de Laghouat' in (2.8) pertains to the expression *lieutenant-colonel Gaillard de Saint-Germain* or if it's identified as a separate entity of a possible *function* type or if only the contained toponym is labelled. Basically, each of the three options represents a conceivable directive.

(2.8) Le lieutenant-colonel Gaillard de Saint-Germain, *commandant supérieur du cercle de Laghouat*, passe du 1er au 4e chasseurs d'Afrique.

A possible NE annotation for the sentence in (2.8) is given in (2.9) with entities put in square brackets. This solution actually applies a simplified version of the set of

categories used for the annotation of the QOP corpus and complies with the respective guidelines. Illustrated by means of the expression *commandant supérieur du cercle de Laghouat*, where a top-level *function* category successively includes *organization* and *location* classes, some annotation standards even demand hierarchically structured or nested NEs.

(2.9) Le [$_{\text{FUNC}}$ lieutenant-colonel] [$_{\text{PER}}$ Gaillard de Saint-Germain], [$_{\text{FUNC}}$ commandant supérieur du [$_{\text{ORG}}$ cercle de [$_{\text{LOC}}$ Laghouat]]], passe du [$_{\text{ORG}}$ 1$^{\text{er}}$] au [$_{\text{ORG}}$ 4$^{\text{e}}$ chasseurs d'[$_{\text{LOC}}$ Afrique]].

In this chapter, historically and OCR conditioned spelling variation has been presented as constituting a factor which requires particular attention when dealing with NLP related tasks in digitized historical texts. Moreover, some basic aspects and considerations with respect to NER have been discussed. Since this thesis focuses on sequence labelling approaches which are based on recurrent neural networks, the relevant theoretical and technical framework is introduced in Chapter 3.

3 NER as a Sequence Labelling Task

This chapter outlines the theoretical and technical foundations regarding the method chosen to approach NER in historical data. As indicated in the previous chapters, the present thesis focuses on NER designed as as sequence labelling problem. This task is tackled with the help of neural sequence classifiers which, for their part, build on neural character-level language models. Correspondingly, the following sections provide, on the one hand, an overview of the sequence labelling task as a whole and, on the other hand, an introduction to the neural architectures which are relevant in the context of this work.

3.1 Sequence Labelling

NER, as well as a variety of other NLP related task, such as part of speech (PoS) tagging and chunking, is commonly approached as a sequence labelling problem. For this purpose, a sentence, or text in general, is treated as the sequence of its tokens. Here and in the remainder of this thesis, the terms *word* and *token* are used synonymously, not only to refer to meaningful language units but also to punctuation marks. For the processing of a sentence considered as the sequence of its tokens, a sequence model is instructed to provide each element in this sequence with a label in order to generate the corresponding label sequence. In doing so, the labels are chosen according to a fixed set of categories, which, in the case of NER, comprises entity types such as *person*, *organization* and *location*.
With the aim of labelling the words in new unseen sentences, a classifier internalized the correspondences between labels and respective tokens which it observed in a quantity of example sequences. To this, the pre-annotated example material used to learn the word-label correspondences has to be formatted in a suitable way. When dealing with NER as a sequence labelling task, the IOB tagging scheme represents a usually resorted format which allows for expressing an entity's type as well as its boundaries (Jurafsky and Martin, 2018, p. 329f.). Hence, while the notation in Example (2.9) makes use of opening and closing square brackets to signal an entities left and right boundaries, the IOB format utilizes, as the naming suggests, the

letters *I*, *O* and *B* in order to fulfil this function. Thus, tokenized, i.e., split into its tokens, and tagged according to the IOB scheme, the first two clauses of the sentence in (2.9) would adopt the annotation format shown in Figure 3.

Tokens	**IOB labels**
Le	O
lieutenant-colonel	B-FUNC
Gaillard	B-PERS
de	I-PERS
Saint-Germain	I-PERS
,	O
commandant	B-FUNC
supérieur	I-FUNC
du	I-FUNC
cercle	I-FUNC
de	I-FUNC
Laghouat	I-FUNC

Figure 3: IOB tagging scheme applied to the noun phrase 'Le lieutenant-colonel Gaillard de Saint-Germain'.

In the IOB notation, the *O* label, which stands for 'outside', is reserved for non-entity words, e.g., the token *Le* in Figure 3. Each word forming part of an NE is annotated, depending on whether it constitutes the start or an inside element of the entity, with either a *B* or an *I* tag and the particular entity type from the set of categories. In this way, the *B* prefix signalizes the beginning of an NE. A label with an initial *B* or *I* followed by another *B*-prefixed label or an *O* tag implicitly indicates the end of an entity. Therefore, the IOB scheme encodes the identical information as the square brackets notation. Moreover, it allows for formatting text and the corresponding NE annotations as sequences of words and labels, respectively (Jurafsky and Martin, 2018, p. 330).

Basically, the implementation of further label layers or, with regard to the representation in Figure 3, the addition of supplementary IOB columns would even allow for considering nested NEs such as the ones exemplified in (2.9). However, the NER task focused in this thesis does not include hierarchically structured entities and, consequently, IOB tagging is applied in accordance with the annotation format exemplified in Figure 3.

Eventually, a line-based text file with tokenized and IOB annotated sentences constitutes a practical format for the example data from which a sequence classifier deduces word-label correspondences. In a classical machine learning scenario, such files with tens of thousands of labelled sentences are used to train a sequence classifier.

Typical state-of-the-art classifiers which are trained on annotated corpora comprise

implementations of feature-based as well as neural sequence models. Commonly used models relying on word pattern, spelling and, optionally, dictionary features include probabilistic maximum entropy Markov models (MEMMs) and linear chain conditional random fields (CRFs). In order to assist these classifiers in the learning of correct word-label correspondences, auxiliary information, e.g., a word's base form or its PoS tag, may be incorporated in the IOB annotated data and accessed as additional features.
On the other hand, the employment of long short-term memory (LSTM) architectures represents the modern neural approach to tackle sequence labelling problems. Actually, a series of sequence classifiers which combine neural models with an on-top CRF component (Chiu and Nichols, 2016; Ma and Hovy, 2016; Akbik et al., 2018) sets the state-of-the-art benchmarks for NER and other NLP tasks.
For the sake of completeness, it is noted that rule and dictionary-based approaches represent an additional alternatives to set about NER. Though, applying fixed patterns to identify expressions in texts which are characterized by spelling variation and randomly introduced noise is deemed rather inefficient. Accordingly, such classifiers are not considered within the scope of this thesis.

3.2 Classical Sequence Modelling

Sequence models learn to predict a word sequence's corresponding label sequence based on the example material they are trained on. Since hardly any training data set sufficiently covers toponyms or person names, unknown words constitute a major challenge if a new sentence is presented to a NE classifier. Typically, feature-based sequence models rely on a series of spelling and context features in order to appropriately handle such unfamiliar instances.
In addition to a word itself, its context window, i.e., a determined number of words to its left and right, can be considered as a feature. On the other hand, spelling features comprise word suffix and prefix spans of adjustable lengths as well as a variety of binary features. With regard to the latter, it may be taken into account, for example, whether or not a word is capitalized, if the contained letters are all upper-cased or not and if there are word internal punctuation marks and/or digits. Furthermore, word pattern features are intended to record the abstract shape of a word. They can be created, for instance, by converting each upper and lower-case character into 'a' and 'A', respectively, and every digit into '0'. If available, a word's base form and PoS tag as well as information about its presence or absence in supplementary person and place name lists can be encoded as possible features.

In the end, the defined features are automatically extracted for all words occurring in the example data and can be accessed for the learning of the word-label correspondences. Likewise, when it comes to the labelling of a new sequence, the same features are created for the words not present in the training set.
Basically, HMMs are possible candidates to approach sequence labelling problems (Jurafsky and Martin, 2018, p. 158–167). Given an observation sequence $O = o_1, ..., o_n$, they model the probability of a sequence of hidden states $H = h_1, ..., h_n$. Thus, analogous to words and entity labels in the NER task, a HMM only sees the observations, whereas the hidden states are not directly visible to the model. Actually, HMMs model the joint probability $P(H, O)$ of associated sequences of observations and hidden states. To this, they base upon the assumption that the probability of a hidden state h_i depends exclusively on the a preceding hidden states $h_{i-a}, ..., h_{i-1}$. Moreover, they assume that the probability of an observation o_i depends solely on the hidden state h_i. Accordingly, HMMs estimate the best hidden state sequence $\hat{H}$ as follows:

$$\hat{H} = \arg\max_{H} \prod_{i=1} P(h_i|h_{i-a}, ..., h_{i-1}) \cdot P(o_i|h_i). \tag{3.1}$$

Hence, in addition to the circumstance that HMMs computes the desired probability $P(H|O)$ by indirect means, i.e., by applying Bayes' theorem, the integration of features presupposes substantial modifications and is restricted to an individual observation o_i, which implies that neighbouring observations may not be considered for the identification of the hidden state h_i (Jurafsky and Martin, 2018, p. 167f.).
On the contrary, maximum entropy Markov models (MEMMs) determine $\hat{H}$ by directly calculating the conditional probability $P(H|O)$:

$$\hat{H} = \arg\max_{H} \prod_{i=1} P(h_i|o_i, h_{i-1}). \tag{3.2}$$

MEMMs (McCallum et al., 2000) enhance HMMs by means of incorporating the logistic regression algorithm, which is consecutively applied to each of the observations in O in order to include the hidden state assigned to the previous observation for the determination of the following observation's hidden state (Jurafsky and Martin, 2018, p. 167f.). According to Equation (3.2), a basic implementation of an MEMM would condition the probability of a hidden state h_i on the observation o_i and the preceding hidden state h_{i-1}. However, MEMMs are able to consider any number of preceding hidden states and an arbitrary amount of observations to the left and right. Moreover, MEMMs can handle any kind of features in form of feature func-

tions. Such binary-valued functions are applied to the previous hidden states and the observation context in order to determine if a feature is present or absent. Each of these feature functions is associated with a weight, which is learned during model training.

Since MEMMs implement successive applications of exponential models in order to estimate the conditional probability of each hidden state h_i, the probability of the sequence H is given by the concatenated outcomes of these separately applied operations. Consequently, MEMMs still miss the potentially useful direct influence of a future hidden state h_{i+1} on the determination of the current hidden state h_i. This unidirectional behaviour of MEMMs' is eliminated by CRF models (Lafferty et al., 2001). According to Lafferty et al. (2001, p. 283), the elementary difference between the two models is that

> a MEMM uses per-state exponential models for the conditional probabilities of next states given the current state, while a CRF has a single exponential model for the joint probability of the entire sequence of labels given the observation sequence. Therefore, the weights of different features at different states can be traded off against each other.

Thus, given a set of feature functions $f_1, ..., f_j$ and the associated weights $\lambda_1, ..., \lambda_j$, a CRF models the joint probability $P(H, O)$ as follows:

$$P(H, O, \lambda) = \frac{1}{Z(O)} \exp\left\{\sum_{i=1}^{n}\sum_{j} \lambda_j f_j(h_{i-1}, h_i, O, i)\right\}, \quad \text{where } Z(O) = \sum_{h \in H}\sum_{i=1}^{n}\sum_{j} \lambda_j f_j(h_{i-1}, h_i, O, i). \tag{3.3}$$

Since CRF models take into account observation context as well as neighbouring hidden states, they prove to be especially suitable for sequence labelling problems in general and, in particular, for the NER task. Actually, there are several freely available NER tools which implement pre-trained linear chain CRFs. These include, for instance, the *Stanford Named Entity Recognizer*[6] (Finkel et al., 2005) for English or *GermaNER*[7] (Benikova et al., 2015) for German texts. Likewise, a series of tool kits, such as *MALLET*[8] (McCallum, 2002) or *Wapiti*[9] (Lavergne et al., 2010), facilitate the training of own CRF models.

[6] https://nlp.stanford.edu/software/CRF-NER.shtml

[7] https://github.com/tudarmstadt-lt/GermaNER

[8] http://mallet.cs.umass.edu

[9] https://wapiti.limsi.fr

3.3 Neural Sequence Modelling

In the preceding section, we have seen that classical sequence models rely on handcrafted context and spelling features. In this section, an alternative method based on neural networks (NNs) is described. Commonly, neural sequence classifiers do not presuppose manually elaborated features to adequately perform a labelling task, but rather learn useful representations automatically. For this thesis, recurrent neural networks (RNNs) or, more precisely, the LSTM architecture is chosen to approach NER in historical texts. This specialized architecture facilitates the processing of arbitrary length input sequences and enables the learning of long-distance dependencies, which is particularly advantageous in the case of the focused NER task. In the following, it is given an overview of the notions and concepts which are necessary to understand NN-based approaches for sequence labelling problems.

3.3.1 Neural Networks

Loosely alluding to the way of how information is received, processed and forwarded in the human brain, a NN can be described as an aggregation of interconnected computational units, each of which is designated to produce a scalar out of an input array of real valued numbers (Jurafsky and Martin, 2018, p. 132). To this array of numbers is associated a weight vector and an additional bias term. Thus, for the actual input of a neural unit, a bias term $\boldsymbol{b}$ is added to the dot product of a vector $\boldsymbol{x}$ and its corresponding weight vector $\boldsymbol{w}$. To the resulting value, the unit applies an usually non-linear function f in order to generate the output scalar h:

$$h = f(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}) \tag{3.4}$$

Established functions to be applied by neural units, referred to as *activation functions*, are the sigmoid, to convert the output into a decimal between 0 and 1, the hyperbolic tangent (tanh), to produce a value ranging from -1 to $+1$, and the rectified linear unit (ReLU) function, which maps negative numbers to 0 and retains positive ones. For example, recurring to the sigmoid function σ, a unit's output would be calculated according to Equation (3.5).

$$h = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}) = \frac{1}{1 + \exp(-(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}))} \tag{3.5}$$

Usually assembling a bunch of computational units, a NN organizes them in at least three stacked layers. In a rather simple specification, these layers correspond to an input, a hidden and an output layer. While not counting the input layer, this basic configuration would be considered a two-layer NN, being the hidden the first and the output the second layer.
Regarding the connections between the different layers, the output of an input layer unit may serve as input for each of the hidden layer units. The output of a hidden layer unit, in turn, can be passed to each of the units of the output layer. Such a network, where the units of each layer, except for the ones of the output layer, provide the input for the units of the next layer, conforms to a feed-forward neural network (FFNN).
The two-layer FFNN drawn in Figure 4 shows a hidden layer with four computational units, each of which takes as input the outputs produced by the three units of the input layer. For instance, the leftmost hidden unit processes x_1, x_2 and x_3 from the input layer and generates h_1, which, for its part, is forwarded to each of the three output layer units. Accordingly, the aggregate of scalars produced by a layer's units can be considered as a list of numbers or, in other words, a vector.

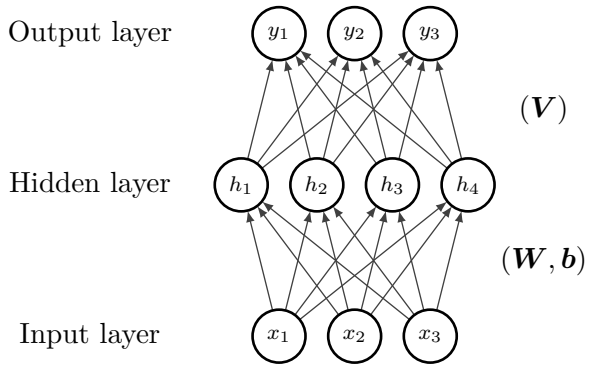


Figure 4: Example of a simple feed-forward neural network with one fully connected hidden layer. The weight matrices $\boldsymbol{W}$ and $\boldsymbol{V}$ are associated to the input-hidden and the hidden-output connections, respectively. The bias term $\boldsymbol{b}$ is associated to the connections between the input and the hidden units.

Given the four units of the hidden layer and the three-dimensional vector $\boldsymbol{x} = [x_1, x_2, x_3]$ resulting from the input layer in Figure 4, the associated weights can be represented as the weight matrix $\boldsymbol{W}$ of dimension 3×4. Hence, the hidden layer's output vector $\boldsymbol{h}$ is generated by applying basic matrix operations, i.e., multiplying the vector $\boldsymbol{x}$ with each matrix column $\boldsymbol{W}_i$. By means of illustration, Equation (3.6) shows the entire operation to calculate the vector $\boldsymbol{y}$ produced by the output layer in Figure 4 (a possible bias term $\boldsymbol{c}$, associated to the connections between the hidden

and output layer units, is omitted in Figure 4 as well as in Equation (3.6)):

$$\boldsymbol{y} = \sigma(\boldsymbol{V} \cdot f(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b})) \tag{3.6}$$

Whereas the sigmoid function is the right choice for binary classification, the function executed on the scalars of the output units usually corresponds to a normalizing softmax function if multinomial classification tasks are approached. This function is intended to convert a list of k real numbers into a probability distribution of k probabilities. For example, the network in Figure 4 can be used to predict the NE type of a word in a sequence of words. After applying the softmax function, the final output $\hat{\boldsymbol{y}}$ encodes a probability distribution over the three categories *person*, *organization* and *location* and the predicted label label is the most probable one.
Eventually, FFNNs, such as the two-layer network shown in Figure 4, are able to assume the part of a classifier that takes decisions based on the features originating from dedicated hidden layers. Here, the hidden layer's activation functions play a key part in the success of NNs. If these functions were exclusively linear ones, this would imply a mere succession of linear transformations of input vectors. Consequently, the actual capacity of NNs to emulate a variety of mathematical functions by alternately stacking linear transformations and non-linear functions would be disregarded (Goldberg, 2017, p. 41–44).
As it is the case for classical sequence models, the final objective of implementing neural sequence classifiers consists of predicting $\hat{\boldsymbol{y}}$ for a given input $\boldsymbol{x}$. This goal, i.e., the minimization of the discrepancy between the network's output $\hat{\boldsymbol{y}}$ and the anticipated correct output $\boldsymbol{y}$, is reached by means of common machine learning techniques. Thus, with the aim of learning the layer associated $\boldsymbol{V}$, $\boldsymbol{W}$ and $\boldsymbol{b}$ parameters, a loss function $l(\hat{\boldsymbol{y}}, \boldsymbol{y})$ is defined to measure the difference between the actual and the desired output. On the other hand, the gradient descent algorithm is applied to identify the minimal loss. The loss function often comes up to the cross-entropy loss used for logistic regression or, in the case of multinomial classification, to the categorical cross-entropy loss. For a multi-layer network with numerous neural units per layer, the amount of parameters to be learned is typically enormous. Accordingly, the computation of the loss function's gradient, the vector indicating the trend of maximum change rate per parameter, requires a strategy specific to neural model training, i.e., the application of the backward propagation of errors algorithm (Jurafsky and Martin, 2018, p. 140–145).
Whereas the $\boldsymbol{V}$, $\boldsymbol{W}$ and $\boldsymbol{b}$ parameters are learned during model training, a bunch of hyperparameters affecting the network's performance have to be set before the learning procedure is started. First of all, the number of hidden layers and com-

putational units per layer as well as the applied activation functions have to be defined. Furthermore, the learning rate, i.e., the adjustment rate of the weights regarding the loss gradient, the number of epochs, or, how many times the entire training material is processed, and the mini batch size, which controls the division of the training data into smaller fragments, represent customizable training preferences. Likewise, drop-out training may be applied by setting drop-out probabilities to, for example, 0.5. This effects that, for each training example, the hidden layer's individual units are zeroed with a probability of 50%. This technique addresses the problem of over-fitting, which refers to the phenomenon of learning the weights in such a way that the final model performs well on the training corpus but generalizes poorly to unseen data (Goldberg, 2017, p. 47). In the end, finding the optimal hyperparameters frequently turns out to be an empirical process which involves giving a try to different configurations and checking them in repeated test runs.

3.3.2 Recurrent Neural Networks

In the previous section, we have seen that FFNNs are capable of learning to predict the $\hat{\boldsymbol{y}}$ given the input $\boldsymbol{x}$. With respect to the processing of word sequences, this kind of NNs shows a drawback analogous to the one of HMMs. Hence, for the labelling of a particular word, only its neighbouring words in the form of a fixed size context window may be considered, which impedes internalizing "useful long-distance relations between words" (Chiu and Nichols, 2016, p. 357). In order to overcome this weakness inherent to FFNNs, RNNs implement a method which allows for considering a broadened context. Actually, this extended context comes up to the entire word sequence.

3.3.2.1 Simple Recurrent Networks

The improvement over FFNNs, which enables RNNs to handle word sequences of any length, is reached by means of a slightly adapted network architecture. To this, the most basic specification of RNNs, which corresponds to the one of simple recurrent networks (SRNs) (Elman, 1990), includes recurrently directed layer outputs. Thus, while the vector produced by an FFNN's hidden layer is only passed to the next layer, an SNR implements a layer which directs its output to a next layer as well as to itself or, more precisely, the neural units of an SRN hidden layer process their own output (Jurafsky and Martin, 2018, p. 177). Figure 5 exemplifies the difference between an FFNN (on the left) and an SNR (on the right), which corresponds to an additional circular flow of the latter's hidden layer output.

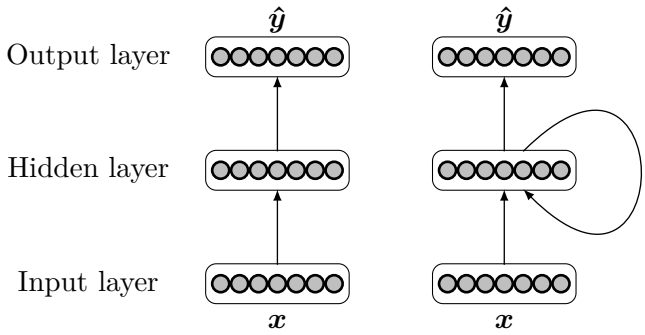


Figure 5: Comparison of a feed-forward neural network (left) and a simple recurrent networks (right). The central difference of the two networks is represented by the simple recurrent network's recurrently directed output of the hidden layer (for illustrative purposes, the layer units' outputs are reduced to one outgoing arrow per layer).

When processing a sequence, the single sequence elements are presented to the network one after another or, in other words, one at a time. Thus, successively provided with new sequence elements, the network generates the corresponding output per time step before proceeding to the next element. Accordingly, Figure 6 illustrates the SRN from Figure 5 unrolled in time.

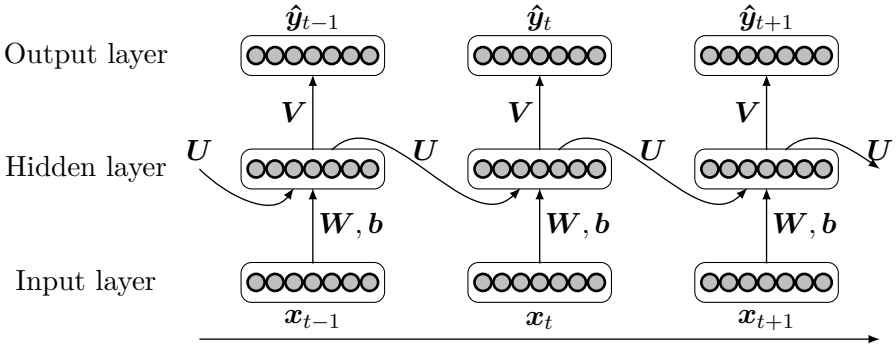


Figure 6: Example of a simple recurrent network unrolled in time. The weight matrix $\boldsymbol{U}$ is associated to the hidden layer's recurrently directed output vector.

Given, for instance, a sequence of words, where the individual words come up to either their respective equivalents from pre-trained word embeddings or randomly initialized vectors, a word representation $\boldsymbol{x}_t$ is fed into the network at time step t. When the hidden units at time step t are next to generate their output, they receive, in addition to the values which produces the input layer based on $\boldsymbol{x}_t$, the output $\boldsymbol{h}_{t-1}$ from the hidden layer at the preceding time step. In turn, the output of the

hidden layer at time step t is forwarded, on the one hand, to the hidden layer at time step $t+1$ and, on the other hand, to the second layer at the current time step, which produces the output $\hat{\boldsymbol{y}}_t$. In order to estimate the SNR's output vector $\hat{\boldsymbol{y}}_t$, the previously introduced operation for FFNNs (Equation (3.6)) is expanded as follows:

$$\hat{\boldsymbol{y}}_t = \sigma(\boldsymbol{V} \cdot f(\boldsymbol{U} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W} \cdot \boldsymbol{x}_t + \boldsymbol{b})) \tag{3.7}$$

In contrast to the calculation of an FFNN's output, the operation shown in Equation (3.7) includes the vector $\boldsymbol{h}_{t-1}$ from the previous time step, which is multiplied with its associated weight matrix $\boldsymbol{U}$. Accordingly, the weights in $\boldsymbol{U}$ regulate how and to what extent the preceding context encoded in $\boldsymbol{h}_{t-1}$ influences the network's output $\hat{\boldsymbol{y}}_t$. Since this modification enables SRNs to recall information from the first up to the current time step, the restriction to a fixed size context window for the prediction of a sequence element's label is eliminated.
Analogous to FFNNs, the weight matrices $\boldsymbol{V}$, $\boldsymbol{W}$ and, additionally, $\boldsymbol{U}$ represent the network's parameter, which are shared across all time steps. Likewise, a loss function $l(\hat{\boldsymbol{y}}, \boldsymbol{y})$ is defined to measure the added up losses from all time steps and backpropagation through time is applied to determine the gradient used for parameter adjustment (Goldberg, 2017, p. 166f.). However, the training of an SRN, or, RNNs in general, actually doesn't focus the assignment of labels to a sequence of input elements. Usually, an SRN rather constitutes "a trainable component in a larger network", where "the final prediction and loss computation are performed by that larger network, and the error is back-propagated through the RNN" (Goldberg, 2017, p. 167). Thus, what makes its integration in a superordinate network interesting for sequence labelling tasks are an SRNs' hidden states, i.e., the vectors produced by the hidden layers. These equal automatically generated features and assist the global sequence classifier in performing the targeted prediction task.

3.3.2.2 Bi-Directional Recurrent Neural Networks

Whereas FFNNs are bound to limited context windows, SRNs are capable of taking into account an entire sequence for the generation of a given sequence element's output. Actually, this applies only in the case of the last sequence element. Since the output $\boldsymbol{h}_{t-1}$ of the hidden layer at the previous time step includes information from the beginning up to the element processed at time step $t-1$, the outputs of the hidden layers at subsequent time steps cannot be accessed when processing the input $\boldsymbol{x}_t$. However, evidence from future time steps may prove quite useful for the calculation of $\boldsymbol{y}_t$.

Considering the sentence in (2.7), for instance, the expression *Saint-Germain* at the beginning of the sequence could be identified as referring to either a place or a person. Here, the subsequent context 'a fait' ('has made'), which is rather to be expected in conjunction with an animate individual than with a city district, allows for excluding the label *location*. Thus, a method that releases the entire left and right context for the labelling of a given sequence element comes up to the implementation of a bi-directional RNN (Bi-RNN). Actually, this solution corresponds to the combination of two separately trained SRNs. Accordingly, the first SRN processes a sequence of n elements from the left to the right and the second network reads the same sequence in the opposite direction. Figure 7 illustrates the basic composition of a Bi-RNN, which comprises a forward and a backward SRN fed with one and the same input sequence.

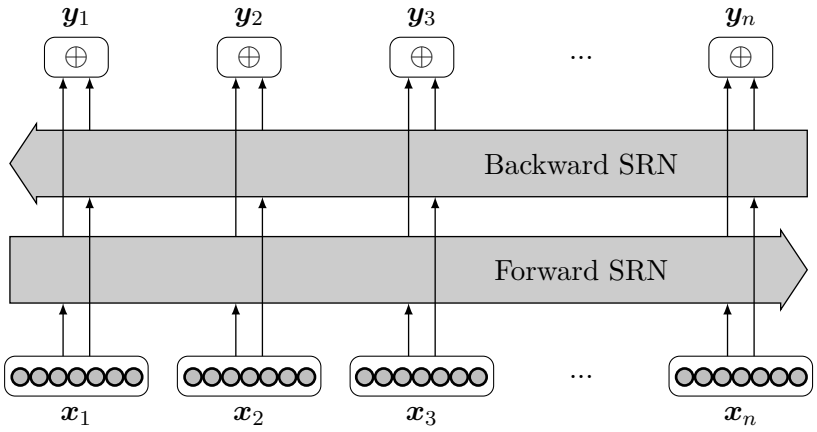


Figure 7: Schematic illustration of a bi-directional recurrent neural network according to Jurafsky and Martin (2018, p. 188).

At a particular time step t, a Bi-RNN has access to the hidden layer output or, in other words, the hidden state $\boldsymbol{h}_t^F$ from a forward SRN as well as to the vector $\boldsymbol{h}_t^B$ from a backward network. Whereas $\boldsymbol{h}_t^F$ encodes information from the first sequence element $\boldsymbol{x}_1$ up to the one processed at time step t, the state $\boldsymbol{h}_t^B$ keeps track of evidence from the last element $\boldsymbol{x}_n$ to $\boldsymbol{x}_t$. The Bi-RNN's hidden state $\boldsymbol{h}_t$ is the concatenation of the vectors $\boldsymbol{h}_t^F$ and $\boldsymbol{h}_t^B$ from the respective SRNs:

$$\boldsymbol{h}_t = \boldsymbol{h}_t^F \oplus \boldsymbol{h}_t^B \tag{3.8}$$

Eventually, the vector resulting from the combination of the forward and backward networks' hidden states at time step t comprises information from the entire left and right contexts, which can be accessed by the overall model in order to produce the output $\boldsymbol{y}_t$.

3.3.2.3 Long Short-Term Memory

Although SRNs are able to handle arbitrary length sequences and to build up context information based upon already processed sequence elements, they turn out to be rather inapplicable in practice. According to the number of time steps to pass through, the calculation of the gradients during network training involves iterative multiplication of the weight matrix $\boldsymbol{W}$ associated to the hidden layer's input vector (Goldberg, 2017, p. 181). As an effect, the gradients tend either to drop to zero or to grow exponentially when longer sequences are processes. While such exploding gradients may be countered by including a training parameter which effects the clipping of gradients above a defined threshold, the problem of gradients moving towards zero is addressed by the LSTM architecture (Hochreiter and Schmidhuber, 1997), a subclass of RNNs specifically designed to avoid vanishing gradients (Goldberg, 2017, p. 60). To this, the LSTM architecture implements special computational units which are used in a network's hidden layers. These units are equipped with "smooth mathematical functions that simulate logical gates" through which new input is passed in order to determine the share of relevant information to be included for future time steps (Goldberg, 2017, p. 180). Likewise, obsolete information is filtered out from the existing memory with the purpose of creating hidden states that come up to only the valuable part of the previous context.

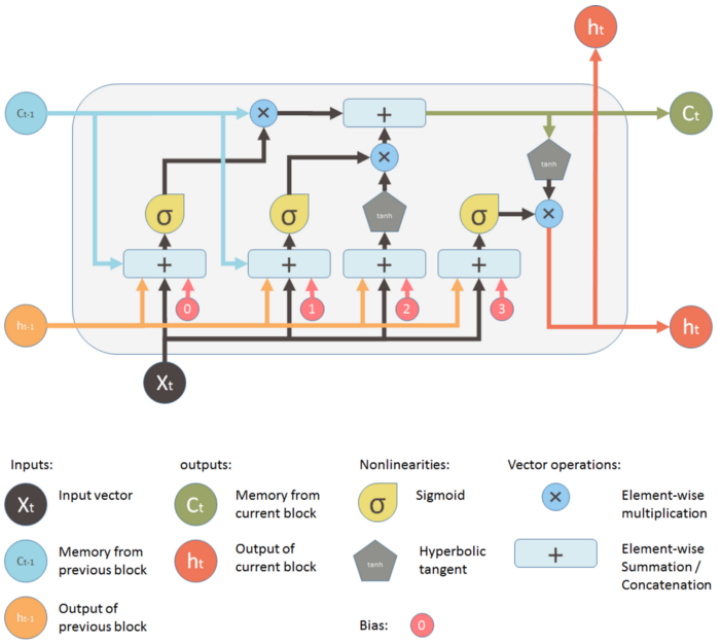


Figure 8: Schematic representation of a basic LSTM unit according to Yan (2016). The sigmoid functions constitute the input, forget and output gates through which the input at time step t as well as the output and the memory from the previous time step $t-1$ are passed.

Figure 8 visualizes a schematic representation of a LSTM unit intended to produce the memory $\boldsymbol{c}_t$ which encodes the relevant history up to time step t. Here, the first two sigmoid functions correspond to the unit's input and forget gates through which the memory $\boldsymbol{c}_{t-1}$ and the output $\boldsymbol{h}_{t-1}$ from the previous time step as well as the current input $\boldsymbol{x}_t$ are passed. The third sigmoid function correlates with the output gate which takes the $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ vectors.
The principle behind LSTM networks is basically the same as the one of SRNs, with the key difference that the former are capable of managing useful context by means of selecting relevant and discarding non-relevant information. Thus, analogous to Bi-RNN, LSTM networks may be implemented as bi-directional (Bi-LSTM) networks as well. With respect to sequence processing, implementing Bi-LSTM networks constitutes the currently favoured approach since it allows for modelling "highly non-trivial long-distance dependencies" by taking "into account an effectively infinite amount of context on both sides of a word" (Chiu and Nichols, 2016, p. 357).

3.4 Neural Language Modelling

The job of a language model (LM) is defined as estimating the probability of a sequence, e.g., a sequence of characters or words, to occur in a particular language. In addition, an LM calculates an element's probability to follow next in the sequence given the previous sequence elements. Recurring to the chain-rule of probability, where each element depends on the respective foregoing elements, the probability of a sequence X of length k is expressed as follows:

$$P(X) = P(x_1)P(x_2|x_1)P(x_3|x_{1:2})P(x_4|x_{1:3})...P(x_k|x_{1:k-1}) \tag{3.9}$$

Classical approaches to estimate the probability of an entire sequence or a next sequence element are based on relative frequency counts of element n-grams. To this, the incidences of, for example, different character trigrams in a large data set are divided by the number of times the respective first two characters are observed together (Jurafsky and Martin, 2018, p. 38–43).
However, the language modelling task can be accomplished, just as well or even better, by NNs. Thus, instead of providing probability distributions over labels, a FFNN may be instructed to output for each input element the corresponding probability distribution over possible next elements (Goldberg, 2017, p. 109–112). Moreover, the hidden states of the LSTM architecture may encode the entire sequence history up to the element to be processes at the current time step. Accordingly, given their

capability to memorize an unlimited number of preceding elements, LSTM networks are highly suitable for the language modelling task.

3.5 Evaluation Metrics

This section specifies the evaluation metrics used for the assessment of the performances of the sequence classifiers which were trained in the context of this thesis. Moreover, it introduces the measurement employed to indicate the quality of an LM.

3.5.1 Sequence Model Performance

The performance of a sequence model is commonly evaluated by means of precision and recall and the harmonic mean of the two measures, referred to as F_1 *score*. These measurements are based on a classifier's prediction results, which can take the following outcomes:

- **True positive** (TP): The classifier correctly predicts the presence of class A, i.e., class A is predicted where there actually is class A.
- **False positive** (FP): The classifier wrongly predicts the presence of class A, i.e., class A is predicted where there actually is class B.
- **False negative** (FN): The classifier wrongly predicts the absence of class A, i.e., class B is predicted where there actually is class A.

Indeed, NER considered as a sequence labelling problem typically does not come up to a binary classification task with only two possible classes. Thus, FP and FN instances in a NER scenario correspond to those cases in which entities are either erroneously identified or missed, respectively. Moreover, an NE may be composed of multiple words. In Section 3.1, we have seen that sequence models which perform a labelling task according to the IOB tagging scheme assign a label to each word in an sentence. Consequently, correct predictions of only a part of an entity are possible. However, such partial matches are not considered for this thesis. Instead, it is followed the definition adopted for the CoNLL-2003 Shared Task, which determines that a "named entity is correct only if it is an exact match of the corresponding entity in the data file" (Tjong Kim Sang and De Meulder, 2003, p. 144). Accordingly, the evaluation script `conlleval.pl` (F.1), which was provided for this shared task, is used to calculate the performance scores of the sequence models trained in the context of the present work.

The initially indicated measures, calculated by means of the TP, FP and FN entity counts for a given data set, are specified as follows:

- The **precision** (P) comes up to the proportion of the identified entities which are classified correctly. It is computed based on TP and FP instances:

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{3.10}$$

- The **recall** (R) corresponds to the proportion of correctly classified entities among all possible entities in the data set. It is calculated based on TP and FN instances:

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3.11}$$

- The $\mathbf{F_1}$ **score** (F_1) combines precision and recall into an evenly weighted harmonic mean. It is calculated with the following operation:

$$\mathrm{F}_1 = 2 \cdot \frac{\mathrm{P} \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}} \tag{3.12}$$

Hence, an NE classifier which achieves a high precision value reliably assigns correct labels to the recognized entities. On the other hand, a high recall implies that the sequence model is able to properly identify most of the entities in the data set. Accordingly, a high F_1 score suggests that the classifier performs both tasks in a satisfactory manner.

3.5.2 Quality of LMs

A possible method to asses the quality of an LM corresponds to its evaluation in terms of perplexity. This measurement indicates how good a model manages the task of predicting a particular sequence of elements (Goldberg, 2017, p. 106). In the case of character-level LMs, the perplexity is calculated based on the probabilities assigned to the n characters $c_1, c_2, ...c_n$ in a text sample. Therefore, given an LM function q which estimates character probabilities on the basis of preceding characters, the character-level perplexity is computed as follows:

$$2^{-\frac{1}{n}\sum_{i=1}^{n} \log_2 q(c_i|c_{1:i-1})}. \tag{3.13}$$

The operation in (3.13) results in low perplexities for high character probabilities. Thus, the better an LM performs the prediction task, the lower the perplexity value.

4 Related Work

This thesis focuses on NER in digitized historical texts which is approached by means of neural sequence models. The combination out of the two essential aspects, i.e., sequence labelling based on NNs and NER in historical texts, appears in only a few contributions so far. This is due to the fact that state-of-the-art NER systems with a central neural component are commonly evaluated on contemporary data. Hence, in order to demonstrate the competitiveness of these models, the indicated performance scores mostly refer to the material provided for dedicated shared tasks, such as CoNLL-2003 or GermEval-2014. On the other hand, NLP for historical texts often faces lack of sufficient amounts of annotated example data, which poses a detaining factor for machine learning approaches, including deep learning methods. For settings characterized by data sparseness, rule-based systems or implementations of linear chain CRF sequence models still represent a reasonable alternative. In view of these circumstances, the overview of previous work presented in this chapter is organized in a twofold manner. Thus, NER in historical texts and NER based on neural sequence models are treated separately in Section 4.1 and Section 4.2, respectively.

4.1 NER in Digitized Historical Data

An essentially classical approach is followed by Crane and Jones (2006), which present the results from their extensive work on identifying NEs in 19th century issues of the *Richmond Times Dispatch*. The core of their system constitute comprehensive gazetteers and other external resources as well as an elaborate set of rules attuned to the used NE categories. Focusing on the historical period of the American Civil War (1861–1865), they apply a rather specific tagging scheme. Thus, besides the conventional *person*, *place* and *time* classes, they introduce categories such as *regiment*, *ship* and *railroad*. The evaluation of the system is carried out manually over a random sample of the most frequently tagged entities. The performance scores are primarily indicated in terms of accuracy in order to assess the correctness of assigned types and established co-references, among other things. Findings from

this manual analysis directly enter the further refinement and enhancement of the patterns used for entity identification and labelling. By means of adjusting rules, they achieve excellent accuracies for some categories, e.g., values of above 90% for *ship* and *regiment* type entities, and an overall accuracy of 85.01%. Consequently, Crane and Jones (2006, p. 38f.) stress the importance of, on the one hand, sophisticated patterns and, on the other hand, carefully selected external knowledge sources that cope with the data's domain and its temporal origin.

Likewise, Grover et al. (2008) rely on a set of rules and external lexical resources for NER in the *Journals of the House of Lords* from the late 17th and early 19th centuries. But, while seemingly of no relevance for the 1860s issues of the Virginian newspaper analyzed by Crane and Jones (2006), the digitized British Parliament publications are reported to show a considerable amount of OCR and OLR errors. In the light of OCR-conditioned errors and historical spelling variation, Grover et al. express doubts concerning the suitability of feature-based sequence models and, therefore, opt for a rule-based approach. In their set-up, which is referred to as a preliminary one, they only consider *person* and *location* type entities. Furthermore, additional rules intended to detect OCR errors and noise are applied in therms of an upstream task. The actual NE patterns are applied separately, i.e., *person* type entities are identified before the *location* NEs in order to prevent faulty tagging of ambiguous entities. Regarding the evaluation on a manually annotated share of the 19th century data, the authors report an overall F_1 score of 71.81 and values of 66.53 for the *location* and 75.60 for the *location* categories. Suggestions for improving the results are largely formulated along the lines of the ones by Crane and Jones (2006). However, since Grover et al. locate the main error source in the noise originating from the data preprocessing, they propose a procedure aimed at automatically enhancing the OCR quality of the data. Thus, a machine learning approach is eventually considered, not for the NER task itself but rather for the correction of miss-recognized characters.

Turning away from previous rule-based approaches, Mac Kim and Cassidy (2015) resort to the *Stanford NER* in order to identify NEs in digitized articles of 19th and 20th centuries Australian newspapers. Although their data is assessed as being of acceptable OCR quality, a noticeable quantity of noise and miss-recognized characters is reported especially for the earlier articles. Given this factor, the authors assume potential performance differences between a model trained on contemporary data[10] and another one trained on OCRed historical data. Accordingly, Mac Kim and Cassidy contrast a pre-trained model and a model trained on in-domain data, which is automatically annotated using the pre-trained Stanford NER tagger, and

[10] *Stanford NLP* provides pre-trained English CRF models which are trained on the English data sets from CoNLL-2003, MUC-6 and MUC-7 shared tasks.

evaluate them on a manually labelled share of the digitized articles. Their evaluation of the two systems reveals that the pre-trained classifier (71 F_1) outperforms the in-domain model (67 F_1). They attribute the latter system's slightly lower performance to the comparatively small amount of training data. Consequently, Mac Kim and Cassidy conclude that incrementing the number of merely 600 newspaper articles used for the training of the in-domain classifier would effect a positive impact on the performance scores.

Another attempt of applying CRF models to digitized historical newspapers is described by Neudecker (2016), who presents the efforts aimed at creating NE annotated training material for researchers interested in the field. These data corresponds to 400 OCRed pages of French, Dutch and German newspapers from the early 17th to the late 20th centuries, which were manually annotated within the context of the *Europeana Newspapers* project[11]. Regarding these documents, Neudecker indicates a sometimes significant portion of recognition errors and a notable amount of spelling variation in the pages dating from before 1900. Resorting to the *Stanford NER*, he trains language-specific sequence models on the manually annotated data and evaluates them in terms of k-fold cross-validation. In the case of the French model, for instance, F_1 scores of 83, 79 and 58 are achieved for the three categories *person*, *location* and *organisation*, respectively. Since Neudecker also supposes that OCR errors and historical spelling variation have an impact on the performance of the trained classifiers, he investigates experimental approaches to tackle faulty annotation attributable to these phenomena. However, artificial contamination of clean entities with spelling errors as well as rule-based correction of spelling variation are discarded due to lacking benefits and/or intricate implementation. In the end, these findings induce the author to state that performance gains would most likely be reached with additional training data and advanced feature engineering.

The two annotated German sets following from the work of Neudecker (2016) are subsequently used in a comparative study by Riedl and Padó (2018). This data comes up to a larger corpus (87,000 tokens) with journals from the Dr. Friedrich Teßmann Library (LFT) and a smaller one (35,000 tokens) containing newspapers from the Austrian National Library (ONB). On this data sets, respectively split into training, development and test portions, *Stanford NER* and *GermaNER* classifiers are evaluate in order to contrast them with two Bi-LSTM-based models. For the latter are used pre-trained word embeddings which are computed either on *Wikipedia* (WikiEmb) or on historical texts from *Europeana* (EuroEmb). With the aim of investigating performance divergences between CRF and neural classifiers trained on different kinds of data, i.e., small, historical sets as opposed to large, contemporary

[11] http://www.europeana-newspapers.eu

corpora, Riedl and Padó likewise include the German data sets from CoNLL-2003 and GermEval-2014 shared tasks. Concerning the CRF and neural systems trained and evaluated on the smaller ONB sets, they indicate significantly lower F_1 scores for the neural models. Trained on the larger LFT sets, CRF and Bi-LSTM models perform pretty much on the same level. In view of the neural systems' tendency to achieve better results when trained on larger amounts of data, the authors propose a transfer learning approach, where training is started on a large-scale contemporary corpus and finished on a smaller in-domain data set.

Model	**Train set**	**Test set**	**F_1 score**
GermaNER	LFT_{train}	LFT_{test}	69.18
	ONB_{train}	ONB_{test}	73.31
Bi-$LSTM_{WikiEmb}$	(GermEval-2014_{train} +) LFT_{train}	LFT_{test}	(74.33) 68.47
	(GermEval-2014_{train} +) ONB_{train}	ONB_{test}	(76.06) 70.46
Bi-$LSTM_{EuroEmb}$	(GermEval-2014_{train} +) LFT_{train}	LFT_{test}	(72.03) 69.62
	(GermEval-2014_{train} +) ONB_{train}	ONB_{test}	(78.56) 67.29

Table 1: Performances comparison of CRF and neural models in the context of the transfer learning experiments conducted by Riedl and Padó (2018). Transfer learning data and results are indicated in brackets.

With respect to the performances of the Bi-LSTM systems, Riedl and Padó ascertain remarkable benefits when applying the transfer learning method. As summarized in Table 1, the neural systems trained on two different corpora achieve consistently higher F_1 scores (displayed in parenthesis) than the equivalent models trained on the respective historical sets only. Moreover, given F_1 values of 73.31 (ONB) and 69.18 (LFT) for the best CRF classifier (*GermaNER*), Riedl and Padó show that neural networks can readily outperform CRF models even if in-domain training data is sparse.

4.2 NER Based on Neural Sequence Models

Huang et al. (2015) evaluate CRF, LSTM and Bi-LSTM models as well as combinations out of the two NNs and the CRF model on different sequence labelling tasks, including PoS tagging, chunking and NER. By applying a CRF model to the output of a Bi-LSTM network, they introduce in the field a system which leverages the neural network's forward and backward states as past and future features and, by means of the CRF component, label information on sentence level. The input

for the Bi-LSTM-CRF model, and likewise for the other networks and model combinations, consists of n-gram and a series of spelling features. Additionally, either random initialized or pre-trained word embeddings are used. With respect to the NER task, the proposed Bi-LSTM-CRF model outperforms the other networks on the English CoNLL-2003 data set. Thus, Huang et al. demonstrate that combining CRF models with RNNs represents an improvement over, for instance, simple implementations of CRF sequence models. However, their approach is essentially dependent on hand-crafted features and external knowledge sources. This becomes apparent when one and the same Bi-LSTM-CRF network is provided with either pre-trained word embeddings only (84.74 F_1) or the ensemble of word, context and spelling features (88.83 F_1). Moreover, using additional dictionary features results in the overall highest performances score (90.10 F_1).

With the aim of reducing the dependence on extensive feature engineering for NER tasks, Chiu and Nichols (2016) introduce a NN based system with a component explicitly dedicated to automatically extract character-level features. This part is assumed by a convolutional neural network (CNN), which is fed with the words' concatenated character embeddings and, optionally, additional feature vectors encoding each character's type, i.e., if it represents an upper or lower cased letter, a punctuation mark or another symbol. The character-level word representations generated by a CNN are thought to reflect the words' morphological information in a similar way as it would be achieved by means of hand-crafted spelling or word shape features. These vectors, combined with the respective vectors from pre-trained word embeddings, are then passed to a Bi-LSTM network. In addition to the character and word embeddings, Chiu and Nichols include supplementary spelling and dictionary features on an experimental basis. These experiments, performed on English CoNLL-2003 data, reveal that the CNN-Bi-LSTM network which is provided with the additional features achieves an F_1 score of 91.62. Thus, since the F_1 for the CNN-Bi-LSTM model which does not employ supplementary features drops by almost eight points, their system still relies on some simple word features and, additionally, external knowledge sources.

Combining the approaches of Huang et al. (2015) and Chiu and Nichols (2016), Ma and Hovy (2016) propose a method that implements a CNN-Bi-LSTM network with an on-top CRF model. However, in contrast to previously introduced neural models, they announce their system as a truly end-to-end one, which neither presuppose any hand-crafted features nor data preprocessing. Like Chiu and Nichols (2016), they resort to CNNs in order to convert words into a representation format appropriate for being fed into an intermediate Bi-LSTM network. Thus, the actual input of the Bi-LSTM network is generated by concatenating the words' character-level representations from a CNN with traditional pre-trained word embeddings. With respect to

the evaluation on the Englisch CoNLL-2003 data set for NER, Ma and Hovy report improvements over the results obtained by a series of previous neural and classical sequence labelling approaches. After all, they show that a Bi-LSTM-based model that neither relies on feature engineering nor on external knowledge is capable of producing state-of-the-art results for NER and other sequence labelling tasks.
The idea of an upstream component designated to automatize feature engineering is also picked up by Lample et al. (2016), who focus on domain independent NER for different languages. Thus, with a view to providing a sequence model with conducive input, they aim at learning word representations that encode character-level as well as word-level information. In doing so, they instantiate an input layer which passes a word's character representations in the form of random initialized character embeddings through an initial Bi-LSTM network. For the final word representations, the per word output of this Bi-LSTM network is combined with the respective vectors from pre-trained word embeddings. Thus, the resulting word representations are thought to model morphological information as well as information on word distribution. Both random initialized character-level and traditional word embeddings constitute model parameters to be learned and fine-tuned, respectively. Whit respect to the actual sequence modelling task, Lample et al. (2016) experiment with two different approaches: a Bi-LSTM-CRF model as implemented by Huang et al. (2015) and Ma and Hovy (2016) and a rather non-orthodox Stack-LSTM network, which combines LSTM with a chunking algorithm derived from transition-based (shift-reduce) parsing methods. Regarding the performance on the English CoNLL-2003 NER data, the authors indicate an F_1 score of 90.94 for the Bi-LSTM-CRF model which relies on the proposed input word representations.
Likewise focusing the generation of appropriate input word representations, Akhundov et al. (2018) present an approach that slightly deviates from the one introduced by Lample et al. (2016). They also instantiate an input layer which includes a Bi-LSTM network but, in contrast to Lample et al., they do not employ word-level character embeddings as underlying units. Instead, words are considered as sequences of their characters' UTF-8 encodings, e.g. `0x62 0x79 0x74 0x65` for the word *byte*. Similar to the input layer of Lample et al. (2016), the LSTM networks' outputs per word are concatenated with the corresponding vectors from pre-trained word embeddings in order to build the final word representations. Thereafter, Akhundov et al. also employ a Bi-LSTM-CRF network intended to perform the actual sequence labelling task. With respect to the English CoNLL-2003 NER task, the neural sequence model which relies on byte embeddings outperforms the analogous model of Lample et al. (2016) by 0.17 F_1.
Introducing another word representation variant, Akbik et al. (2018) claim to reach new state-of-the-art F_1 scores on English and German CoNLL-2003 NER data sets.

They present so-called *contextual string embeddings*, which are derived from the hidden states of neural character-level language models. Thus, an upstream Bi-LSTM network, trained to predict the next character in a sequence, assigns a forward-backward hidden state to each character in a sequence. These states encode character-level information of an entire sequence. Consequently, the proposed embeddings represent a word as the concatenated hidden states of its constituting characters. According to the authors, this allows for modelling "words and context fundamentally as sequences of characters, to both better handle rare and misspelled words as well as model subword structures such as prefixes and endings" (Akbik et al., 2018, p. 1639). Indeed, the obtained results seem to confirm the effectiveness of their approach. Using contextual string embeddings as input for a Bi-LSTM-CRF model, alternatively combined with common word and/or character embeddings, Akbik et al. report astonishing F_1 scores of up to 93.09 on English CoNLL-2003 NER data.

In Table 2 are shown the F_1 scores achieved by the respective best performing systems, evaluated on English CoNLL-2003 NER data, from the above introduced LSTM-based sequence labelling approaches. Although all of them report results of above 90 F_1, it can be stated that the simple combination out of a Bi-LSTM network and an on-top CRF model represents the dominant and most successful one. The tendency to completely omit manual feature engineering and to leave this process to dedicated upstream processes also becomes apparent. Accordingly, these upstream tasks or, to put it another way, the representation formats generated by them constitute the aspect in which the individual Bi-LSTM-CRF implementations most notably differ.

Model	**F_1 score**
Huang et al. (2015)	90.10
Lample et al. (2016)	90.94
Akhundov et al. (2018)	91.11
Ma and Hovy (2016)	91.21
Chiu and Nichols (2016)	91.62
Akbik et al. (2018)	93.09

Table 2: Results for different LSTM-based sequence models evaluated on English CoNLL-2003 NER data.

Bearing in mind the data focused in this thesis, the contextual string embeddings introduced by Akbik et al. (2018) seem to represent a promising sequence model input since they are announced as being especially robust with regard to spelling errors and rare words. It is supposed that NER in digitized 19th century newspaper

texts, which is characterized by occasional outdated spelling variants as well as considerable amounts of OCR-conditioned noise and errors, would benefit from this embedding type. Therefore, the approach followed in the present work corresponds to a Bi-LSTM-CRF model based on contextual string embeddings, which are treated in more detail in Section 5.2.

5 Data, Method and Tools

In the present thesis, NER in digitized historical texts is approached as a sequence labelling problem. The historical corpus used for this work corresponds to the *Quaero Old Press* corpus, which had originally been created from OCRed French newspaper documents. As indicated in Section 4.2, the neural approach chosen to tackle the NER task includes contextual string embeddings as proposed by Akbik et al. (2018). Accordingly, the following sections delineate, on the one hand, the data on which the focused sequence labelling task is performed and, on the other hand, the applied neural classifiers as well as the employed word representations. In addition, the computational tools which were utilized to train the neural sequence and language models are presented.

5.1 Data

The central data basis of this thesis comes up to an NE annotated French newspaper corpus which had been established in the course of a sub-project of the European *Quaero* programme[12]. In this section, the original corpus as well as the set of entity categories and the guidelines according to which the data had been annotated are described. Likewise, the necessary preprocessing steps, namely tokenization and conversion to the IOB format, are specified. These procedures, carried out in the context of the present work, were necessary to prepare the data for the focused sequence labelling task.

5.1.1 The *Quaero Old Press* Corpus

As indicated by its originators, the *Quaero Old Press* (QOP) corpus[13] comprises 76 issues of the three French journals *Le Temps*, *La Croix* and *Le Figaro* (Galibert et al., 2012, p. 3127). Each of these newspaper issues, which are preserved as

[12]http://www.quaero.org/

[13]http://catalogue.elra.info/en-us/repository/browse/ELRA-W0073/

digital images at the National Library of France, dates from December 1890 and assembles an average number of four pages. By means of a combined optical layout and character recognition tools, the 295 digital documents were converted into machine-readable text and subsequently cleaned from non-relevant elements, such as captions, charts or advertisements. Hence, the QOP data actually comes up to a collection of OCRed articles from nearly 300 historical newspaper pages.
Following the preliminary digitization and cleaning processes, human annotators were requested to perform the manual NE labelling task on the remaining OCR output. This annotation campaign was performed in accordance with the *Quaero* annotation guidelines (Rosset et al., 2011), which had been elaborated in the course of previous *Quaero* projects. Eventually, the annotations produced by the participating annotators were included in the form of XML-like elements. Accordingly, in the annotated text, each word or multi-word expression identified as an NE is preceded and followed by a start and an end-tag, respectively. By this means, the boundaries as well as the categories of the occurring entities are encoded. For illustrative purposes, an extract from a QOP corpus file is shown in Figure 9.

```
138
139  00212636/PAG_2_TB000032.png
140  côte.
141
142  00212636/PAG_2_TB000033.png
143  » Les <pers.coll> colons, </pers.coll> dont ii est commode de médire,
144  sont peut être un pou bruyants, mais avec quelle
145  vaillance ils font grandir en <loc.adm.nat> Algérie </loc.adm.nat> le nom de la
146  <loc.adm.nat> France. </loc.adm.nat>
147
148  00212636/PAG_2_TB000034.png
149  » Dans la présente discussion on n'a ;ut;ra
150  abordé que des points de détail, et l'on n'a pas
151  touché aux questions générales.
152
```

Figure 9: Example of a QOP data file (extract) which contains entity annotations in the form of XML-tags.

On the one hand, the lines in the provided QOP files are relatively short. This is due to a layout characteristic of the OCRed newspaper documents, which organize the contents in rather small columns. As a consequence, the individual article lines hardly ever correlate with complete sentences and often end or start with hyphenated words. On the other hand, the lines in the annotated files appear grouped into blocks of variable sizes. These blocks, separated from each other by empty lines, correspond to the OCR outputs for a number of smaller image segments into which the original newspaper pages had been divided by the OCR tool. An example of such a page section is shown in Figure 10. Actually, the text in this image correlates with the second line block in Figure 9.
Since the division of the documents does not exhibit particular regularities, the lines grouped together rarely conform to complete paragraphs or entire newspaper

articles. Moreover, in some extreme cases, a block is made up of only one line which contains a single word (and possibly a final punctuation mark). This applies, for instance, to the first block in Figure 9. Here, a paragraph's last line, which is made up of only one word, had been processed as a separate image.

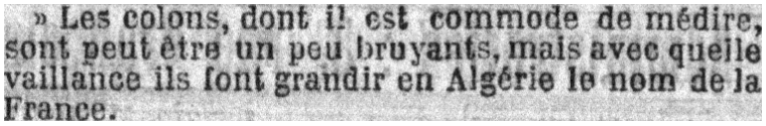


Figure 10: Example of an image used for OCR preprocessing in the context of the digitization of the QOP corpus.

As shown in Figure 9, each block is preceded by a line which specifies the path where the respective image file is located. To this, an eight-digit number is used to refer to the relevant newspaper issue and the name of the PNG file indicates the page and the image segment in question. According to the number of included documents, the QOP corpus provides a total of 295 data files which contain the annotated OCR output per newspaper page. Since this data was used in a *Quaero* evaluation campaign on NER (Galibert et al., 2012), the corpus files are already split into training and test portions. Table 3 shows the sizes of the QOP test and training sets in terms of number of document files, lines and tokens.

	Train set	**Test set**	**Total**
Documents (pages)	231	64	295
Lines	192,543	61,088	253,631
Tokens	1,297,742	363,455	1,661,197

Table 3: Sizes of the QOP training and test data set according to Galibert et al. (2012, p. 3129).

5.1.1.1 The *Quaero* Annotation Scheme

The annotation guidelines used by the human annotators of the QOP corpus were initially elaborated during an annotation and evaluation campaign which aimed at the creation of a factual data bank from resources associated with the news realm (Grouin et al., 2011, p. 92). Since they were first applied to transcribed speeches from francophone broadcasting stations (Grouin et al., 2011, p. 96), the *Quaero* guidelines presupposed an NE annotation scheme that encompassed the categories relevant for the news domain. Thus, in addition to the conventional categories *person*, *location*, *organization*, *time* and *amount*, Grouin et al. (2011) included the

types *function* and *product* and introduced subtypes for nearly all of the defined main categories. The resulting set of 7 types and 30 subtypes is shown in Table 4.

Types	*amount*	*func*	*loc*	*org*	*pers*	*prod*	*time*
Subtypes		*.ind*	*.adm.town*	*.adm*	*.ind*	*.object*	*.date.abs*
		.coll	*.adm.reg*	*.ent*	*.coll*	*.doctr*	*.date.rel*
			.adm.nat			*.art*	*.hour.abs*
			.adm.sup			*.serv*	*.hour.rel*
			.phys.geo			*.rule*	
			.phys.hydro			*.media*	
			.phys.astro			*.fin*	
			.fac			*.soft*	
			.oro			*.award*	
			.add.phys				
			.add.elec				

Table 4: *Quaero* entity types and subtypes according to Grouin et al. (2011, p. 94). This listing excludes the subtypes *unknown* and *other*, which can be used for all types except for *amount*.

The above listing omits the subclasses employed to label those entities whose subtype is not determinable (*unknown*) or not covered by the tag set (*other*) since they can be applied in combination with most of the main classes. Except for the *amount* category, for which no subcategory is defined, only the types' respective subtypes are used for the actual annotation of NEs.

Indeed, it is not the number or elaborateness of the considered types and subtypes what makes the NE definition proposed by Grouin et al. (2011) notably different from other taxonomies. By contrast, a distinctive aspect of the *Quaero* annotation scheme is represented by the inclusion of so-called *components*, which are designed to classify an entity's constituents (Grouin et al., 2011, p. 94). In this manner, transverse components, such as *name*, *kind* and *qualifier*, can be applied to the constituents of different types of NEs. On the other hand, specific components are restricted to the elements of entities of a particular type. For instance, a *title* component can only be used for the constituents of *pers.ind* (individual person) NEs and the *address-number* component is limited to the NEs of the *loc.add.phys* subtype.

In the rather basic case illustrated in Example (5.1), the expression *socialistes allemands* ('German socialists') is considered an NE of the subtype *pers.coll* (group of people). The two constituents, for their part, are identified as a *qualifier* component, which specifies an NE by a qualifying adjective, and a *demonym* component, which indicates an entity's geographical provenance.

(5.1) <pers.coll> <qualifier> socialistes </qualifier> <demonym> allemands </demonym> </pers.coll>

Hence, NEs as conceived by Grouin et al. (2011, p. 94) are compositional two-level constructs, where types and subtypes are intended for the first annotation layer and components for the second one. In addition, the *Quaero* guidelines allow for nesting entities. To this, category and subcategory labels may be assigned to the constituents of an NE, just as it is done in the case of entity elements being labelled as components. For instance, the entire expression *ministre de la justice* ('minister of justice') in Example (5.2) is globally annotated with the *func.ind* (individual function or profession) subtype and the constituent *justice* receives the *org.adm* (administrative organization) label.

(5.2) <func.ind> <kind> ministre </kind> de la <org.adm> <name> justice </name> </org.adm> </func.ind>

(5.3) un arrangement avec la <org.adm> <loc.adm.nat> <name> France </name> </loc.adm.nat> </org.adm>

A special case of nested NEs is represented by the annotation of metonymically used expressions. In Example (5.3), the word *France* is assigned the label *loc.adm.nat* (country), i.e., the subtype to which the entity intrinsically belongs. However, the context reveals that *France* actually refers to the country's government. Therefore, the NE is additionally annotated with the *org.adm* subtype. In such cases, the *Quaero* guidelines prescribe that the category which covers an entity's metonymic meaning always has to assume the outermost annotation level.

5.1.1.2 QOP-Specific Annotation Directives

In order to cope with the errors introduced by the OCR preprocessing, the annotators of the QOP corpus had to follow a slightly adapted version of the original *Quaero* annotation guidelines (Galibert et al., 2012, p. 3128f.). These adaptations comprise the inclusion of a supplementary *noisy-entities* component, which indicates the presence of an NE in a particular character span. Hence, this component is applied in the case of the sequence 'S"ad.à-MeMouchet,' in Example (5.4). Given an entity occurrence in a character span with missing word boundaries, the entire sequence is annotated with the respective entity type and, additionally, the *noisy-entity* component. For instance, the combination of tags in (5.4) denotes that the labelled span contains a *pers.ind* type entity, i.e., the subsequence 'MeMouchet,', as well as tokens ('S"ad.à-') which are not part of the NE.

(5.4) <pers.ind correction="S'ad. à Me Mouchet,"> <noisy-entities> S"ad.à-MeMouchet, </noisy-entities> </pers.ind>

(5.5) <loc.adm.town correction="Potsdam"> <name> Potsdàîrr </name> </loc.adm.town>

On the other hand, the adapted guidelines integrate an auxiliary *correction* attribute. In the case of the QOP annotation campaign, this adaptation entailed an additional task for the annotators of the corpus. Thus, if an entity affected by faulty OCR was encountered, they had to resort to the additive attribute in order to record the corrected variant of the erroneous character sequence. Correspondingly, the *correction* attribute of the *loc.adm.town* (district or town) tag in Example (5.5) holds the rectified version of the misspelled span 'Potsdàîrr'. Likewise, the corrected equivalent of the defective sequence in Example (5.4) is captured by the respective entity type tag's *correction* attribute.
As indicated in Table 5, the annotators of the QOP corpus managed to record corrections for 6,370 entities and 242 components. These numbers suggest that less than 5% of the total of 148,620 identified entities exhibit miss-recognized characters and/or noise introduced by OCR preprocessing.

	Train set	**Test set**	**Total**
Entities	113,591	35,029	148,620
Entities corrected	4,122	2,248	6,370
Components	166,151	38,495	204,646
Components corrected	204	38	242

Table 5: Entity and component frequencies in the QOP corpus according to Galibert et al. (2012, p. 3129)

It seems that further manual correction effort has been performed since the publication of the results from the QOP annotation campaign by Galibert et al. (2012). An inspection of the annotated NEs reveals that the corpus version used for this thesis comprises 10,897 entity tags and 281 component tags with a *correction* attribute. However, an increased quantity of corrections applies only to the QOP training data and not to the test set, for which the amounts of corrected entities and components are identical to those indicated in Table 5. On the contrary, the test set shows a decrease in the total number of contained entities (32,996). For some reason, the QOP data contains less NEs but more entity corrections than indicated by Galibert et al. (2012). Accordingly, the actual proportion of misspelled entities in the corpus amounts to nearly 7.5%.

5.1.1.3 OCR Quality

With respect to the share of miss-recognized characters in the QOP documents, Galibert et al. (2012, p. 3128) refer a median raw OCR quality rate of 82% and Rosset et al. (2012, p. 42) report a character error rate (CER) of 5.09% and a word error rate (WER) of 36.59%. The value specified by Galibert et al. probably comes up to the OCR accuracy rate, which indicates the proportion of characters for which the recognition is considered unambiguous. This score is quite difficult to reproduce since it is computed by the OCR tool itself.
On the other hand, Rosset et al. (2012) do not provide any information regarding the reference data against which they compare the QOP documents' misspelled and noisy texts. However, the quoted CER may be approached with the help of the recorded corrections for the annotated NEs. Thus, for each entity in the QOP corpus, the CER can be calculated by means of the Levenshtein distance. This metric measures the difference between two strings in terms of character substitutions, insertions and deletions, i.e., how many of theses operations have to be executed in order to convert one string into another. For instance, given the two character sequences 'Potsdàîrr' and 'Potsdam' from Example (5.5), four operations are necessary to transform the first string into the second one. These operations comprise the substitution of 'à' and 'î' by 'a' and 'm', respectively, and the deletion of the two final r characters. Divided by the length of the reference string, this results in a CER of 57.14% for the erroneous sequence 'Potsdàîrr'.
It has already been stated that approximately 7.5% of the annotated NEs in the QOP corpus exhibit OCR-conditioned errors. Considering only these entities, i.e., the NEs with an existing *correction* attribute, the CER per NE averages to 19.32%. By contrast, if all entities in the QOP corpus are taken into account, the average CER amounts to merely 1.44%.

5.1.2 The IOB-Formatted QOP Corpus

In Section 3.1, we have seen that the IOB notation is a commonly used tagging format for NER devised as a sequence labelling problem. Though, the QOP data is provided as raw OCR output in which the annotated NEs take the form of XML elements. Moreover, the *Quaero* annotation scheme is intended to represent NEs as hierarchical constructs with multiple annotation levels, which reduces its suitability for sequence labelling tasks (Dinarelli and Rosset, 2012, p. 1267). Consequently, the preprocessing steps performed in the context of this thesis included, on the one hand, the tokenization of the original QOP data in order to enable the generation of text

files with one token per line. On the other hand, the nested *Quaero* annotation had to be transferred to the simpler IOB tagging format. To this, the decision was taken that only the outermost XML start and end-tags were considered for the creation of the IOB-formatted QOP (QOP_{IOB}) corpus. Therefore, entities inside other entities as well as *Quaero* components were ignored completely.

5.1.2.1 Preprocessing and Conversion to IOB Format

Given the OCR and layout-conditioned line blocks in the QOP data files, the restoration of complete sentences represented an additional preprocessing step. For this purpose, adjacent lines were joined (with a blank space in between) according to the following principles:

- A line beginning with a token which starts with a lower-case letter is appended to the preceding word sequence if the previous sequence does not end with a upper-case letter or a terminal punctuation mark (full stop, exclamation mark or question mark).
- A line beginning with a token which belongs to an entity is appended to the preceding word sequence if the previous sequence does not end with a upper-case letter or a terminal punctuation mark.
- A line beginning with a token which contains more upper than lower-case letters is appended to the preceding word sequence if the previous sequence starts with a token which contains more upper than lower-case letters.

The application of the above operations to the original corpus data sometimes resulted in token sequences which corresponded to more than one sentence. However, it was not attempted to further segment these sequences into separate sentences. Here, a variety of different abbreviations with a final period and other cases of tokens ending in a terminal punctuation mark, which often came up to a wrongly recognized character, impeded the employment of a reliable segmentation method. The actual tokenization process was performed on the word sequences obtained from the sentence restoration step. To that end, it was resorted to the rule-based multilingual tokenizer *Cutter*[14] (Graën et al., 2018), which provides support for French-language input sentences. This tool relies on language-specific abbreviation lists in order to distinguish between abbreviation associated periods and terminal full stops. Accordingly, the most frequent abbreviations in the QOP data were added to the list for the French language. Though, due to a rather unmanageable variety

[14]https://pypi.org/project/cutter-ng/

of abbreviations, which appeared further intensified by reason of erroneous OCR, it was accepted that the extended abbreviation list was far from being complete. Consequently, the QOP_{IOB} data exhibits, amongst other potential tokenization errors, a certain amount of wrongly split abbreviation tokens.
Eventually, the tokenized sentences were converted into IOB format. For this purpose, each token belonging to an NE received an IOB label composed of either a *B* or an *I* tag and the respective original *Quaero* entity label. The non-entity tokens, on the other hand, were provided with simple *O* tags. Figure 11 illustrates the resulting labelling format by means of the example sequence used in Figure 3.

Tokens	**IOB labels**
Le	O
lieutenant-colonel	B-func.ind
Gaillard	B-pers.ind
de	I-pers.ind
Saint-Germain	I-pers.ind
,	O
commandant	B-func.ind
supérieur	I-func.ind
du	I-func.ind
cercle	I-func.ind
de	I-func.ind
Laghouat	I-func.ind

Figure 11: IOB tagging scheme with *Quaero* types applied to the noun phrase 'Le lieutenant-colonel Gaillard de Saint-Germain'.

The entire preprocessing and conversion procedures were executed with the help of the script `quaero2iob.py` (S.1), which takes as input the QOP data files and outputs a single file with IOB-formatted sentences. Actually, the 231 training and the 64 test set files of the QOP corpus were processed separately and, therefore, the initially created QOP_{IOB} corpus assembled a large training and a smaller test file. Though, in order to comply with a classical machine learning set-up, an additional development set was generated by splitting the IOB-formatted test data in two parts. Thus, the final QOP_{IOB} corpus is made up of a training set, which corresponds to the data from the QOP training files, and a test and a development set compiled from the QOP test data.[15]

5.1.2.2 Corpus Statistics

As a result of the data preprocessing and the non-consideration of inner entities, the amounts of tokens and annotated NEs in the QOP_{IOB} corpus deviate from those

[15] The concrete repartition of the QOP test files over the QOP_{IOB} test and development sets is specified in Appendix A, Table 23.

indicated for the original QOP corpus. Hence, the approximately 1.6 million tokens stated for the QOP corpus compare to more than 1.8 million tokens in the QOP_{IOB} data. The discrepancy between the counts for the two corpora is probably due to the circumstance that the number specified by Galibert et al. (2012, p. 3129) is calculated based upon white space separated tokens. In contrast, the quantity quoted for the QOP_{IOB} corpus includes punctuation marks and function words affected by elision, e.g., the article *l'* or the pronoun *j'*, as individual tokens. The exact token counts for the QOP_{IOB} data sets are shown in Table 6.

	Train set	**Dev set**	**Test set**	**Total**
Tokens	1,457,444	204,986	203,345	1,865,775
Entities	103,304	15,262	14,555	133,121
Entities corrected	8,609	1,246	982	10,837

Table 6: Token and entity frequencies in the QOP_{IOB} corpus

Disregarding inner NEs reduces the total number of entities by approximately 9% and the amount of entities containing OCR-conditioned errors remains more or less the same. Thus, the entire QOP_{IOB} corpus reveals an 8% proportion of entities with an available corrected variant. The raw entity counts for the training, development and test set are listed in Table 6.

	Frequency			
Entity type	**Train set**	**Dev set**	**Test set**	**Total**
amount	12,985	1,533	1,716	16,234
func	17,633	3,094	3,133	23,860
loc	18,464	2,140	2,369	22,973
org	9,810	1,505	1,412	12,727
pers	28,102	4,730	3,638	36,470
prod	4,575	617	612	5,804
time	11,735	1,643	1,675	15,053

Table 7: Entity frequencies per entity type and data set in the QOP_{IOB} corpus.

With respect to the individual entity types, the QOP_{IOB} corpus exhibits rather unbalanced frequency counts. For instance, a total number of 36,470 *pers* type NEs is opposed to merely 5,804 *prod* entities. In terms of relative figures, this comes up to proportions of 27.40% *pers* NEs and 4.36% *prod* type entities. While these two categories represent the by far most and least often occurring entity types, the frequencies of the NEs belonging to the other classes range between 12,727 (9.56%)

and 23,860 (17.92%). Though, the distribution over the three data sets appears to be fairly even for the entities of the different categories. Thus, between 70% and 80% of the NEs of each type are attributable to the training set and the remaining 20% to 30% are evenly distributed over the development and the test data. Table 7 shows raw frequency counts per entity type and QOP_{IOB} data set.
In view of the focused sequence labelling task, the amount of training examples based on which the entity predictions are learned as well as the complexity of the NEs impact a classifier's performance on unseen data. In general, it is assumed that the longer an NE, i.e., the more words the entity contains, the more difficult it becomes for a sequence model to identify all relevant tokens as belonging to one and the same entity. Regarding this aspect, there may be observed notable differences between the entities of the seven categories. For example, 80% of the frequently occurring *pers* type NEs in the QOP_{IOB} corpus are composed of one or two words. By contrast, almost 40% of the comparatively sparse *prod* entities exhibit a number of tree or more tokens. Figure 12 shows for each entity type the proportion of NEs which are made up of one, two, tree, four and five and more tokens.

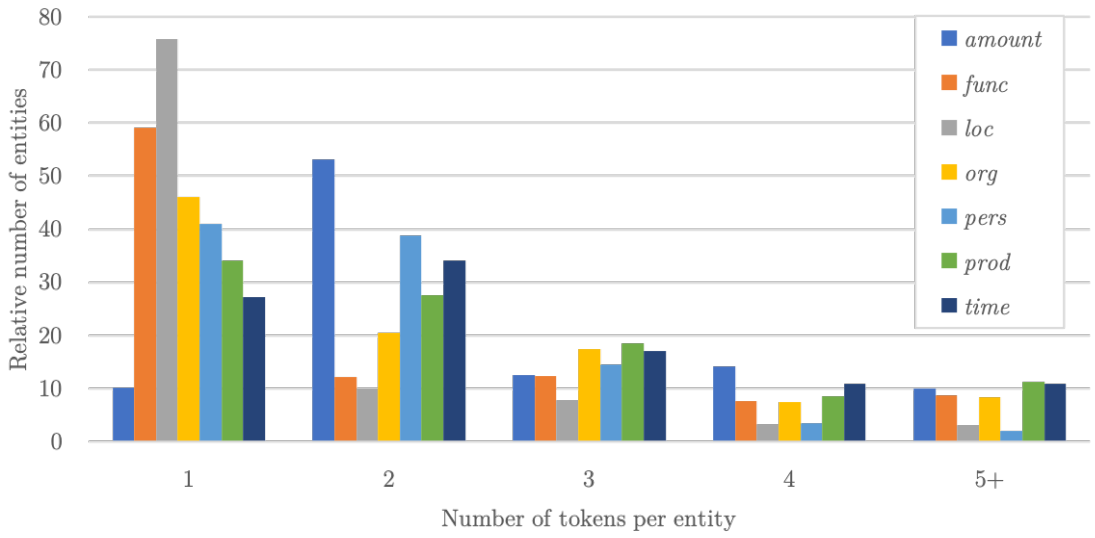


Figure 12: Lengths (number of tokens) of the entities in the QOP_{IOB} corpus.

Given the highest amount of training examples and a vast majority of entities composed of two or less tokens, it is expected that a sequence classifier performs best for the NEs of the *pers* category. As to the *prod* class, the combination out of a small number of training instances and a relatively complex composition of the entities in question presumably leads to inferior prediction results. Other candidates for which a classifier achieves potentially moderate performance scores may be identified in the *amount*, *org* and *time* type entities. In these cases, the proportions of NEs with

three or more tokens amount to approximately 35%. However, the respective numbers of examples are twice or nearly three times the amount of *prod* type entities in the training data.

5.2 Method

In Section 4.2, we have seen that Bi-LSTM-based models offer a valuable alternative when dealing with NER formulated as a sequence labelling problem. Moreover, the many of the recently proposed neural sequence classifiers, mostly following the established design of a Bi-LSTM network combined with an on-top CRF layer, compete on a rather high performance level. Though, with regard to the data presented in the previous section, the Bi-LSTM-based classifier introduced by Akbik et al. (2018) appears to be particularly promising and is therefore adopted for the NER task described in this thesis. Since the chosen classifier makes use of a special embedding type derived from neural character-level LMs, its successful implementation requires training various Bi-LSTM-based models which serve different purposes. Thus, forward and backward neural LMs, trained to predict the next character in a sequence, constitute the basis for the word representations used as input for a Bi-LSTM-CRF sequence model, which, for its part, takes on the actual NE prediction task. Accordingly, the following two subsections are aimed at delineating, on the one hand, the particular embedding type generated by means of character-level LMs and, on the other hand, the neural sequence classifier which makes use of them.

5.2.1 *Flair* Embeddings

Contextual string embeddings, or *Flair* embeddings, are introduced by Akbik et al. (2018) as "a novel type of contextualized character-level word embedding" which exhibits the capability to

> (1) pre-train on large unlabeled corpora, (2) capture word meaning in context and therefore produce different embeddings for polysemous words depending on their usage, and (3) model words and context fundamentally as sequences of characters, to both better handle rare and misspelled words as well as model subword structures such as prefixes and endings.

Whereas the first two points respectively apply to traditional and to contextualized word embeddings as well, the aspect of particular interest for representing words from OCRed historical texts is identified in the third characteristic. Since those

words frequently contain miss-recognized characters and show historically conditioned spelling variation, a representation format that adequately copes with such irregularities is considered highly desirable.

When deploying *Flair* embeddings as word representations, the goal of modelling "words and context fundamentally as sequences of characters" is reached by means of LSTM-based character-level LMs. Such character-level models follow the same principles as the neural LMs introduced in Section 3.4. Though, instead of words being the atomic units of language modelling, a sentence or text is considered as the sequence of its containing characters and, consequently, a LM is trained to predict the next character in the sequence. Thus, at each time step, the model holds a hidden state which encodes the context from the beginning of the sequence up to the respective character. With the aim of providing information from the right-sided context, i.e., from the end of the sequence up to the character at a particular time step, a second LSTM network reads the same character sequence in reverse direction. In Figure 13, the whole set-up is displayed in terms of a Bi-LSTM network with the forward component coloured in red and the backward one shown in blue.

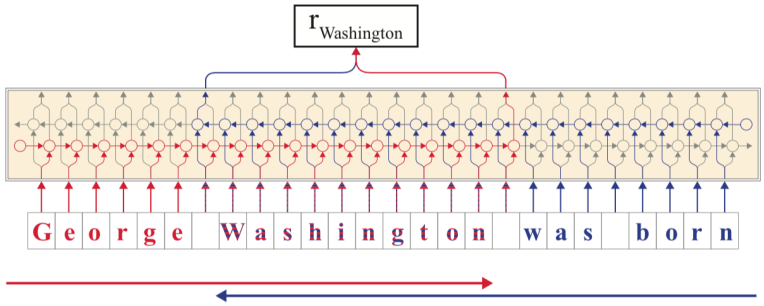


Figure 13: Schematic representation of the extraction of a *Flair* embedding by means of a Bi-LSTM network (Akbik et al., 2018, p. 1639).

Eventually, the character-level LSTM models' hidden states are used to create word-level embeddings for the words in the input sequence. To this, the forward hidden state following a word's last character and the backward hidden state preceding its first character are concatenated in order to produce the corresponding contextual string embedding. Hence, the final embedding is assembled from hidden states which are calculated based on the characters of the word itself and, more importantly, the characters of the left and right context. In Figure 13, this is exemplified by means of the embedding $\boldsymbol{r}_{\text{Washingtion}}$ for the expression *Washington* occurring in the phrase 'George Washington was born'. Here, the relevant forward and backward hidden states encode the character sequences 'George Washington' and 'Washington was born', respectively. Thus, given an input character sequence with indices $t_0, t_1, ..., t_n$, the embedding vector $\boldsymbol{r}_i$ of a word w_i starting at index t_i is calculated as follows

(the superscript F is used to denote the forward hidden state $\boldsymbol{h}_t$ and B to indicate its backward equivalent):

$$\boldsymbol{r}_i := \begin{bmatrix} \boldsymbol{h}^F_{t_{i+1}-1} \\ \boldsymbol{h}^B_{t_i-1} \end{bmatrix} \tag{5.6}$$

In the end, the application of *Flair* embeddings presupposes pre-trained forward and backward character-level LMs. If not available, they have to be trained on huge amounts of unlabelled data, which requires considerable computation time in order to obtain acceptable LM perplexities. The three forward-backward LM pairs trained in the context of this thesis are described in Section 6.2.

5.2.2 Bi-LSTM-CRF Sequence Models

Besides the representation format of the input words, the neural sequence classifier employed for the NER task does not essentially differ from other recently introduced RNN-based models. Hence, the classifier by Akbik et al. (2018) largely corresponds to the Bi-LSTM-CRF architecture proposed by Huang et al. (2015), which likewise reappears in the models by Ma and Hovy (2016), Lample et al. (2016) and Akhundov et al. (2018). According to Akbik et al. (2018), this basic Bi-LSTM-CRF model is amplified by putting in front of it the Bi-LSTM network described in the previous section. As illustrated in Figure 14, the generation of contextual string embeddings from a given input sentence, i.e., in the form of a sequence of characters, can be considered as an upstream procedure for the actual sequence labelling task.

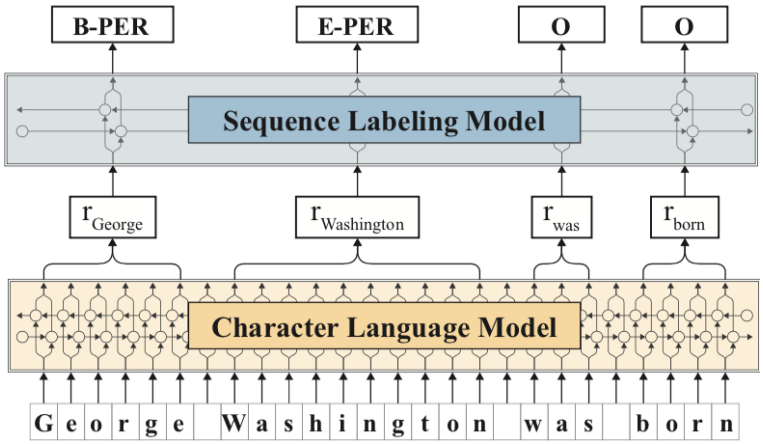


Figure 14: Bi-LSTM-CRF sequence model with *Flair* embeddings as input according to Akbik et al. (2018)

Highlighted in blue in Figure 14, the architecture of the sequence labelling model

implements a Bi-LSTM network which is fed with the character-level LM's output sequence $\boldsymbol{r}_{0:n}$. Based on this input, the network produces the corresponding sequence of combined state vector pairs to which a CRF layer is applied in order to create the desired sequence probability over potential sequence labels. According to Akbik et al. (2018), Equations (5.7) and (5.8) show the operations to calculate the probability of the label sequence $\boldsymbol{y}_{0:n}$ given the character language model's output word vector sequence $\boldsymbol{r}_{0:n}$. To this, the previous label y_{i-1} as well as the current label y_i are taken into account:

$$\hat{P}(\boldsymbol{y}_{0:n}|\boldsymbol{r}_{0:n}) \propto \prod_{i=1}^{n} \psi_i(y_{i-1}, y_i, \boldsymbol{r}_i) \tag{5.7}$$

$$\psi_i(y', y, \boldsymbol{r}) = \exp(\boldsymbol{W}_{y',y}\boldsymbol{r} + \boldsymbol{b}_{y',y}) \tag{5.8}$$

Here, and likewise in the illustration from Figure 14, the sequence classifier's input matches the contextual string embedding sequence $\boldsymbol{r}_{0:n}$ given by the upstream character-level LM. However, a Bi-LSTM-CRF models' underlying input representation may be made up from diverse embedding types. To that effect, stacked embeddings are created by means of concatenating for each input word its vector equivalents from different embedding spaces. Ideally, each resulting word vector combines the representation capabilities of the considered types of embeddings. Hence, while *Flair* embeddings are intended to represent words based on their characters and the surrounding characters, traditional word embeddings are potentially more suitable when it comes to the modelling of semantic and syntactic relations on the word level. Following Akbik et al. (2018), the Bi-LSTM-CRF model adopted for this thesis include stacked embeddings as an alternative to pure contextual string embeddings.

With respect to the assembling and training of Bi-LSTM-based classifiers, and neural character-level LMs as well, Akbik et al. (2018) provide the designated NLP library *Flair*. This framework, to be introduced in Section 5.3.1, constitutes the principal tooling for the compilation of the neural sequence and language models employed for the present work.

5.3 Tools

This section gives an overview of the most important tools and computational resources for the experiments regarding NER based on neural sequence classifiers. In

addition to the formerly mentioned *Flair* framework, employed for the preparation of contextual string embeddings as well as for the training of Bi-LSTM sequence models, further NLP utilities are used to train complementary word representations and feature-based CRF models for comparison purposes. These utilities come up to, on the one hand, the *fastText* library (Bojanowski et al., 2017) for the learning of traditional word embeddings and, on the other hand, the *Wapiti* tool kit (Lavergne et al., 2010), which is devised to train probabilistic models for sequence labelling tasks.

5.3.1 *Flair*

Flair is announced as "a very simple framework for state-of-the-art NLP" which builds on the programming language *Python*[16] (version 3.6 or higher is required) and the machine learning oriented library *PyTorch*[17] (Akbik et al., 2018).[18] Established by the originators of *Flair* embeddings, *Flair* includes contextual string embeddings as a central element. *Flair* embeddings as proposed by Akbik et al. (2018) can be either employed as pre-trained embeddings or trained with the help of the utilities provided by the framework. Thus, regarding the training of Bi-LSTM-based classifiers which rely on contextual string embeddings, *Flair* may be considered an end-to-end solution since it includes the necessary tools to build neural sequence and language models from scratch.

With respect to the training of character-level LMs, *Flair* offers three central classes. In the first place, a `TextCorpus` class loads the example material from which the character sequence probabilities are learned. This corpus of plain text data needs to be previously split into a large training and smaller test and development sets. Depending on its actual size, the training set may be subdivided into several smaller portions in order to avoid memory issues. Second, a `LanguageModel` class is used to instantiate a LM. This step includes the definition of the network's architecture. For instance, an `is_forward_lm` parameter, set to either `True` or `False`, determines whether a forward or a backward model is trained and the parameters `nlayers` and `hidden_size` are aimed at specifying the depth of the network, i.e., how many hidden layers are employed, and the number of computational units per hidden layer. The third class, a `LanguageModelTrainer` class, is appointed to initialize the training of the network. At this, the model's hyperparameters are defined. These include,

[16] https://docs.python.org/3.6/index.html

[17] https://pytorch.org

[18] The *Flair* framework was released by Akbik et al. (2018) in the course of the introduction of *Flair* embeddings and is still under active development. For the experiments performed in the context of the present thesis, the framework's current version 0.4.1 has been employed.

for example, the length of the input sequences (`sequence_length`), the amount of simultaneously processed training instances (`mini_batch_size`) and the number of times the learning algorithm sees the entire training data (`max_epochs`). Similarly, the optimization algorithm and the loss function represent customizable training parameters. Here, the stochastic gradient descent (SGD) algorithm and the cross entropy loss are chosen by default.
Analogous to the procedure for LMs, *Flair* provides three dedicated classes for the training of sequence classifiers. Thus, representing the equivalent to the `TextCorpus` class, a `TaggedCorpus` class allows for loading IOB annotated training material. Likewise, the partition of this data into training, development and test sets has to be carried out beforehand. Thereafter, a `SequenceTagger` and a `ModelTrainer` class are employed to instantiate the sequence model and initialize the network training, respectively. Except for the forward-backward option, the parameters and hyperparameters to be chosen at these steps largely coincide with the ones for LM training. Here, the outstanding difference consists in the definition of the embedding types to be used by a sequence classifier. This is done at model instantiation and requires previously imported embeddings. In the case of applying contextual string embeddings, *Flair* allocates a `FlairEmbeddings` class for loading character-level LMs. Similarly, a `WordEmbeddings` class is intended to import traditional pre-trained word embeddings and a `StackedEmbeddings` class is used to stack different embedding types.
In order to obtain satisfying character-level perplexities, the training of neural LMs typically requires large plain text corpora which comprise hundreds of millions of words. Thus, iteratively working through the examples from such large-scale training material constitutes a time-consuming procedure. Computing *Flair* embeddings based on character-level LMs likewise represents a resource-intensive task, especially if the available data for the training of a sequence model is of substantial size. To this, executing the training of a model on graphics processing units (GPUs), for which the calculation of *Flair* embeddings is actually optimized, helps to noticeably speed up the entire procedure.

5.3.2 *fastText*

fastText[19], a machine learning library written in the programming language *C++*[20], is designed to learn word embeddings which represent each word as the sum of it's character n-grams or, more precisely, the vectors linked to these n-grams (Bojanowski et al., 2017, p. 137). Given a corpus of plain text data, the training of

[19]https://fasttext.cc
[20]https://isocpp.org

fastText embeddings using the the skip-gram NN architecture (Mikolov et al., 2013) is executed by means of the command-line interface:

```
$ ./fasttext skipgram -input corpus -output model
```

To this, the paths of the training corpus and the final model represent mandatory options. Further optional arguments include, for example, the number of times the entire training data is processed, the applied loss function and the dimension of the word vectors. In addition, multi-threading may be employed in order to speed up the learning procedure on modern multi-core central processing units (CPUs). Finally, *fastText* outputs two files, the actual model with its parameters and the vectors for all n-grams and words, on the one hand, and a listing of the word vectors computed for the training data's vocabulary, on the other. Thus, applied to new data, the trained *fastText* model facilitates the creation of word vectors for possible out-of-vocabulary words.

5.3.3 *Wapiti*

Wapiti[21] is an easy-to-use tool kit which enables the training of discriminative sequence classifiers, such as MEMMs and linear chain CRFs (Lavergne et al., 2010). For the training of a sequence model, *Wapiti* presupposes an annotated corpus and a user-defined feature template. The training material has to be written in a data file where each token of the tokenized example sequences occupies its own line and sequences are separated from each other by means of empty lines. Thus, while a sequence word figures at the beginning of a line, supplementary information, like a word's base form or its PoS tag, may be appended consecutively. In any case, the last position of each line is reserved for the label of the word in question. The required feature template, on the other hand, comes up to a pattern file in which are encoded the features to be extracted for model training. Eventually, the training process is started via command-line interface:

```
$ wapiti train -a l-bfgs -p patterns corpus model
```

While *Wapiti* trains a CRF model by default, the input, i.e., the annotated corpus, and the output, which conforms to the trained model, need to be specified. The `-a` and `-p` options are included to indicate the optimization algorithm and the

[21] https://wapiti.limsi.fr

pattern file, respectively. In order to avoid computing the error rate on the training corpus, an additional file with development data may optionally be incorporated by means of a `-d` option. In the end, the output file resulting from model training can subsequently be applied to new word sequences for the purpose of generating the respective label sequences.

6 Experiments and Results

This chapter describes the experiments which were executed in order to determine the Bi-LSTM-based model set-up which performs best on the NER task. For these experimental set-ups, the neural sequence model outlined in Section 5.2.2 was considered with and without an on-top CRF layer and different character-level LMs were used to generate input *Flair* embeddings. Further experimental configurations comprise stacked embeddings which incorporate supplementary *fastText* word vectors, learned from either contemporary *Wikipedia* articles or mixed historical and modern texts. Accordingly, the following sections are intended to introduce the neural language and sequence models as well as the baseline CRF classifiers which were trained in the context of this thesis.

6.1 Baseline CRF Models

The CRF models against which the Bi-LSTM-based sequence classifiers are compared were trained with the help of the *Wapiti* tool kit presented in Section 5.3.3. To this, a pattern file with a series of basic spelling features was created. The considered features include, for example, a word's three-character prefix and suffix, the casing of the initial character and the presence or absence of punctuation marks and digits.[22]

The baseline models were trained on the QOP_{IOB} training set. Additionally, the QOP_{IOB} development data was included for parameter tuning during training and the default limited-memory BFGS algorithm was used for model optimization. By means of repeated training procedures, a total of five CRF classifiers were created. Evaluated on the QOP_{IOB} test set, the best baseline model achieves fairly moderate performance scores of 69.41% recall, 56.23% precision and 62.13 F_1.[23]

[22] The employed feature template is shown in Appendix C.1.

[23] The individual F_1 scores for the five CRF models are indicated in Appendix A, Table 24.

6.2 Neural Character-Level LMs

In order to assess their impact on the overall performance of a Bi-LSTM-based sequence classifier, *Flair* embeddings were calculated on the basis of different neural character-level LMs. Here, difference regarding the used LMs basically relates to the example material on which the models were trained. For the experimental set-ups, in-domain, out-of-domain and mixed-domain data was considered for LM training:

- The **in-domain corpus** corresponds to a collection of digitized French newspaper articles. In addition to the material from the QOP corpus,[24] the contained documents comprise 19th and early 20th centuries issues of the *Journal de Genève*, the *Gazette de Lausanne* and *L'Impartial*. These texts, which had been digitized in the context of the *impresso* project[25], resemble the QOP data in the sense that they, on the one hand, are attributable to a historical French-language newspaper landscape and, on the other hand, exhibit OCR-conditioned spelling variation and random noise. More concretely, the incorporated documents only cover the period from 1870 to 1909[26], which corresponds to a time window of 20 years before and after the publication date of the newspaper issues of the QOP corpus.
- The **out-of-domain corpus** was compiled from a 2018 French *Wikipedia* dump[27]. Accordingly, its content may be attributed to the reference work genre and is assumed to be written in a clean modern standard language.
- The **mixed-domain corpus** equals the joined in-domain and out-of-domain corpora. It therefore represents a mixture of genres, i.e., the reference book and the news genres, as well as a merging of, on the one hand, texts characterized by historically and OCR conditioned spelling variation and, on the other hand, articles composed in a modern standard language with consistent orthography.

Previous to the LM training, each of the three plain text corpora had been randomly sorted and split into a large training and smaller development and test portions. Given training sets of quite considerable sizes, they had additionally been divided into smaller sub-portions in order to prevent memory issues during the training processes. Thus, the training sets of the smaller in-domain and out-of-domain corpora

[24] With a size of merely 1.66 million tokens, the QOP corpus actually constitutes only a minor fraction of the entire in-domain data (500 million tokens).

[25] https://impresso-project.ch/

[26] While this time span correctly applies to the included documents from the JDG and the GDL, the IMP issues merely cover the period from 1881 to 1909. This is due to the fact that the latter newspaper was only founded in 1881.

[27] https://dumps.wikimedia.org/

comprise 21 and 26 splits, respectively, and the larger mixed-domain training data assembles 46 sub-portions. For the three corpora, the count of white space separated tokens per data set is indicated in Table 8.

	Number of tokens		
Data set	**In-domain**	**Out-of-domain**	**Mixed-domain**
Test	4,183,994	4,073,409	4,423,568
Dev	4,306,061	4,043,614	4,477,683
Train	501,272,149	598,899,447	1,107,877,423

Table 8: Sizes of the in-domain, out-of-domain and mixed-domain data sets indicated in number of white space separated tokens.

Equivalent to the number of different training corpora, a total of three forward-backward LM pairs were trained. Eventually, resorting to the utilities provided by the *Flair* framework described in Section 5.3.1, model training was executed using the default SGD algorithm and identical hyperparameters for all LMs. Thus, largely following the recommendations by Akbik et al. (2018, p. 1644), the models are defined as one-layered networks with 2048 hidden states and the chosen hyperparameters include a sequence length of 250, a mini batch size of 100, an initial learning rate of 20, clipping gradients at 0.25 and a maximum number of 75 epochs. Moreover, the annealing factor was set to 0.25 and the patience value to half of the respective number of training splits. These two parameters are intended to regulate the reduction of the learning rate. For instance, given an annealing factor of 0.25 and a patience of 15, the learning rate would be reduced to a fourth of its current value if the training loss doesn't decrease for 15 consecutive training portions. Finally, drop-out training was not considered and drop-out probabilities were accordingly set to 0.[28] The full parametrization can be found in the script `train_cm.py` (S.2), which was used to train the LMs.

Actually, the annealing factor and patience parameters as well as the maximum number of epochs turned out to be irrelevant. On the one hand, the training of each of the LMs was aborted after one week, which in no case allowed for iterating more than six epochs. On the other hand, given the time limit of seven days per LM, the training loss never stagnated for enough trainings splits in order to make the annealing factor take effect. In the end, time constraints and restricted availability of allocatable GPUs[29] prevented the calculation of potentially more sophisticated

[28] For their experiments, Akbik et al. (2018, p. 1644) set drop-out probabilities to 0.25. However, A. Akbik (personal communication, December 21, 2018) states that drop-out training can be considered superfluous if enough training data is available.

[29] 12 GB *GeForce GTX TITAN X*.

character-level LMs (cf. Akbik et al., 2018, p. 1644).
After the one-week training on the respective data, the final LMs show character-level perplexities of between 2.31 for the forward in-domain model and 2.60 for the backward out-of-domain model. Table 9 lists the obtained values for all of the three forward-backward LMs pairs.

	Perplexity		
LM	**In-domain**	**Out-of-domain**	**Mixed-domain**
Forward	2.57	2.31	2.55
Backward	2.60	2.35	2.58

Table 9: Character-level perplexities of the trained forward and backward LMs.

Since character-level LMs are trained to predict a character based on the preceding ones, they may be used to generate arbitrary character sequences (cf. Goldberg, 2017, p. 112). To this, the *Flair* framework provides a `generate_text` method intended to produce such random sequences. By this means, arbitrary character sequences were created based on the final in-domain, out-of-domain and mixed-domain forward LMs. The resulting texts are shown in (6.1), (6.2) and (6.3), respectively.

(6.1) Le parc national du Steauan (abrégé en "Callooides") est un parc situé sur les rives du Rio Grande Blanco. Le mont ne se trouve qu'à l'extrême-nord de l'État de Minas Gerais.

(6.2) LAUSANNE, 2 mars 1906 Uu bel abord, 9 D'urifiants en"Hons ému l'on dans la Salle industrielle. L'expédition est plus pénible encore ; de son côté, ce seront les hommes un moment devant > e détromper,

(6.3) Habilité à la Haarech à un magasin d'étain ayant été installé à Berne et laitière, peut être utilisée pour l'entretien de la fosse. En 1963, elle rejoint la propriété de la (actuelle) d'UtsubursK.

The character sequence in Example (6.1), created with the help of the out-of-domain forward LM, actually constitutes two quite well-formed and grammatical French-language sentences. Moreover, it reflects the explanatory nature of an entry in a reference work, which comes up to the writing genre a *Wikipedia* article belongs to. In contrast, the text generated by means of the in-domain forward model, which is shown in Example (6.2), is quite ungrammatical or incomprehensible and exhibits a couple of odd character combinations as well as badly positioned punctuation marks. While the spelling irregularities and the noise most probably mirror the presence of OCR errors in the training data, there are still sub-sequences assessable as being

characteristic of the news genre. For instance, the initial sequence 'LAUSANNE, 2 mars' corresponds to an element which frequently precedes newspaper articles or short news items. For the text in Example (6.3), produced by the mixed-domain forward LM, it would be quite difficult to decide whether it pertains to the encyclopaedia or to the news genre. This is not least due to the fact that the resulting words, although spelled correctly in most of the cases, combine in a fairly nonsensical way. However, artificial creations such as 'UtsubursK' evidence that the model had been exposed to non-standard orthography during training.

6.3 Neural Sequence Models

The Bi-LSTM architecture described in Section 5.2.2 allows for various configuration options, which include the deployment of *Flair* embeddings alone or in combination with other embedding types as well as the application or non-application of an on-top CRF layer. Thus, experimental set-ups were aimed at investigating the effects of the respective input word representations and the CRF component on the classifier's overall performance. In the context of this thesis, the following six network and embedding type constellations were considered:

- **Bi-LSTM$_{\mathbf{F}}$**: The basic set-up corresponds to the pure Bi-LSTM network that employs *Flair* embeddings as input word representations. Instead of a CRF component, this model uses a softmax classifier as its last layer.
- **Bi-LSTM$_{\mathbf{F+W:out}}$**: The extension of the basic configuration relies on stacked embeddings, for which *Flair* embeddings and word vectors from French *fastText* embeddings are concatenated. The latter may be loaded with the help of the utilities provided by the *Flair* framework. As these embeddings are pre-trained on *Wikipedia* data, they are indicated as out-of-domain word embeddings.
- **Bi-LSTM$_{\mathbf{F+W:mix}}$**: Alternatively, stacked embeddings are produced by concatenating *Flair* embeddings with word vectors from *fastText* embeddings trained on combined *Wikipedia* and historical newspaper data. These embeddings are referred to as mixed-domain word embeddings since they are trained on the mixed-domain corpus presented in the previous section.
- **Bi-LSTM-CRF$_{\mathbf{F}}$**: Another extension of the basic set-up conforms to the full Bi-LSTM-CRF model, i.e., the network with an on-top CRF layer. This configuration makes use of *Flair* embeddings only and therefore matches the model illustrated in Figure 14.

- **Bi-LSTM-CRF$_{\mathbf{F+W:out}}$**: Stacked embeddings are considered for the Bi-LSTM-CRF model as well. For this set-up, *Flair* embeddings are concatenated with out-of-domain word embeddings.
- **Bi-LSTM-CRF$_{\mathbf{F+W:mix}}$**: Analogous to the stacked embeddings alternatives for the basic Bi-LSTM model, a Bi-LSTM-CRF network relying on joined *Flair* and mixed-domain word embeddings is included as well.

Regarding the architecture and the training of the Bi-LSTM and Bi-LSTM-CRF networks, the parametrization was largely left in the *Flair* framework's default setting for sequence tagging models. Hence, the neural sequence classifiers are specified as one-layered networks with 256 hidden states. Model training is performed using the standard SGD optimization algorithm and the applied hyperparameters comprise a mini batch size of 32, an initial learning rate of 0.1, an annealing factor of 0.5, a patience of 3 and a maximum number of 50 epochs. In addition, the default variational drop-out of 0.5 (see Goldberg, 2017, p. 183) is left unchanged. The entire parametrization can be found in the script `train_tagger.py` (S.3), which was employed for the training of the Bi-LSTM-based sequence models.
Concerning the generation of *Flair* embeddings, which are included equally in the case of all considered configurations, the character-level LMs presented in the previous section were used. Therefore, given the in-domain, out-of-domain and mixed-domain LMs, each of the six experimental set-ups was implemented thrice, i.e., once for each of the three forward-backward model pairs. Moreover, it was taken into account that reporting results based on singularly performed experiments is potentially misleading since the repeated training of one and the same NN may lead to varying performance scores (Reimers and Gurevych, 2017, p. 338f.). As asserted by Reimers and Gurevych (2017, p. 338), this is due to the fact that

> the training process for neural networks is highly non-deterministic as it usually depends on a random weight initialization, a random shuffling of the training data for each epoch, and repeatedly applying random dropout masks.

Thus, the experiments conducted in the context of this thesis comprise a series of five repetitions for each of the combinations out of, on the one hand, the three LM pairs and, on the other hand, the six network and embedding type configurations. The neural sequence models were trained on the QOP_{IOB} corpus presented in Section 5.1.2. The performance scores of the single experiments, evaluated on the QOP_{IOB} test data, are calculated by means of the F-measure. Accordingly, the final performance score per repetition series is given by the mean over the F_1 values resulting from the respective five model evaluations. Table 10 lists the obtained F_1

means for all possible experimental set-ups.[30]

	F_1 mean and standard deviation		
Model	**LMs:in**	**LMs:out**	**LMs:mix**
Bi-LSTM$_{F}$	69.05±0.21	68.59±0.20	69.10±0.23
Bi-LSTM$_{F+W:out}$	69.53±0.17	68.79±0.11	69.41±0.35
Bi-LSTM$_{F+W:mix}$	69.64±0.12	69.02±0.14	69.38±0.26
Bi-LSTM-CRF$_{F}$	72.79±0.16	72.55±0.15	73.18±0.20
Bi-LSTM-CRF$_{F+W:out}$	**73.63**±0.22	72.91±0.32	73.39±0.21
Bi-LSTM-CRF$_{F+W:mix}$	73.57±0.39	73.10±0.14	73.39±0.15

Table 10: Performance scores for the different Bi-LSTM-based sequence models which rely on *Flair* embeddings. The F_1 score per model/LMs combination is given by the mean over the F_1 scores which result from the evaluations of five trained sequence models.

As shown in Table 10, the models from the Bi-LSTM-CRF set-ups invariably outperform their Bi-LSTM equivalents by an average of 4 F_1 points. On the other hand, employing in-domain *Flair* embeddings comes up to the best solution for all but the Bi-LSTM$_{F}$ and the Bi-LSTM-CRF$_{F}$ set-ups. In these two cases, the highest performance scores are achieved by using *Flair* embeddings based on the mixed-domain LMs. Therefore, the mixed-domain *Flair* embeddings represent the best choice if classical word embeddings are not available for stacking different embedding types. The employment of stacked embeddings exhibits fairly differential impacts on the respective models' performance scores. Thus, concatenating out-of-domain or mixed-domain *Flair* embeddings with out-of-domain word embeddings leads to slight improvements of up to 0.31 points over the F_1 score means of the corresponding set-ups which solely include *Flair* embeddings. By contrast, using in-domain *Flair* and word embeddings for the Bi-LSTM$_{F+W:out}$ and Bi-LSTM-CRF$_{F+W:out}$ set-ups results in major improvements of 0.48 and 0.84 points compared with the F_1 score means of the respective Bi-LSTM$_{F}$ and Bi-LSTM-CRF$_{F}$ models. Generally, employing mixed-domain instead of out-of-domain word embeddings doesn't make a great difference. If concatenated with mixed-domain or in-domain *Flair* embeddings, mixed-domain word embeddings even bring about a performance decrease in some cases. For instance, the Bi-LSTM-CRF$_{F+W:mix}$ models with in-domain *Flair* embeddings achieve an F_1 score mean of 73.57, which is beaten by the performance scores of the Bi-LSTM-CRF$_{F+W:out}$ models with the same *Flair* embeddings. Actually, this latter set-up, i.e., a Bi-LSTM-CRF network which relies on stacked in-domain *Flair* and out-of-domain word embeddings, proves to be the most successful one.

[30]The individual F_1 scores for the different neural models are indicated in Appendix A, Table 24.

7 Discussion

This chapter surveys the results produced by the sequence classifiers delineated in Chapter 6. From the five trained CRF models, the best performing one is chosen to establish the baseline against which the best Bi-LSTM-CRF$_{\text{F:in+W:out}}$ network and, to a minor extent, the neural models from the other experimental set-ups are compared. In Section 6.3, we have seen that the networks which include in-domain *Flair* embeddings bring about the best performance scores. According to this, the discussion of the results is narrowed to those neural classifiers which generate *Flair* embeddings based on in-domain LMs. The NNs to be contrasted with the CRF classifier are selected analogous to the baseline model. Thus, from each of the six fivefold experimental set-ups, the best performing model is chosen to be included in the performance comparison. With the meaning of a preliminary overview, the performance scores of the CRF baseline model and the neural classifiers which rely on in-domain *Flair* embeddings are listed in Table 11.

Model	**P**	**R**	**F$_1$**
CRF	69.41	56.23	62.13
Bi-LSTM$_{\text{F:in}}$	69.30	69.41	69.36
Bi-LSTM$_{\text{F:in+W:out}}$	70.28	69.11	69.69
Bi-LSTM$_{\text{F:in+W:mix}}$	70.42	69.21	69.81
Bi-LSTM-CRF$_{\text{F:in}}$	75.97	70.37	73.06
Bi-LSTM-CRF$_{\text{F:in+W:out}}$	76.16	71.95	74.00
Bi-LSTM-CRF$_{\text{F:in+W:mix}}$	76.80	71.71	74.17

Table 11: Precision (P), recall (R) and F_1 scores achieved by the best performing CRF classifier (baseline) and the best neural models which rely on in-domain *Flair* embeddings.

Since the present thesis examines potential improvements regarding the identification and labelling of misspelled NEs, a main focus of the subsequent discussion lies on the assessment of how well the considered classifiers perform for such noisy entities. However, before passing to the presentation of the sequence models' concrete output in Section 7.1.2, their overall performances are contrasted and it is provided

an overview of how well they work for the different entity types of the employed *Quaero* tagging scheme.

7.1 Model Performances

As shown in Table 11, all neural classifiers which rely on in-domain *Flair* embeddings clearly outperform the baseline CRF model by 7 up to 12 points F_1. While the basic Bi-LSTM$_{F:in}$ network achieves an F_1 value of 69.36, the employment of stacked embeddings brings about slight improvements of 0.3 to 0.45 points and replacing the Bi-LSTM networks' last softmax layer with a CRF layer leads to additional performance gains of roughly 4 points F_1. Actually, these observations coincide with what has been stated for the different experimental set-ups evaluated in Section 6.3. However, the comparison of the scores listed in Table 11 reveals that the Bi-LSTM-CRF network which relies on stacked in-domain *Flair* and mixed-domain word embeddings performs best (74.17 F_1). Nevertheless, since the repeated training and evaluation procedures yielded a higher F_1 mean for the Bi-LSTM-CRF$_{F:in+W:out}$ models (see Table 10), the most successful classifier from this latter set-up is referred to as the overall best performing one.

A main drawback of the baseline classifier is that it correctly identifies only 8,184 of the 14,555 NEs in the test set, which results in a fairly modest recall of 56.23%. In this respect, the neural models show a much better performance. For instance, the best neural classifier properly labels 10,473 entities, which comes up to a recall of 71.95%. Contrasting the values of the different NNs, though, reveals that possible improvements range within less than 3% recall. On the other hand, the baseline model's precision (69.41%) is quite close to the ones of the Bi-LSTM networks without an on-top CRF layer, or even better if compared to the value of the Bi-LSTM$_{F:in}$ candidate (69.30%). This may be due to the circumstance that the CRF classifier exhibits a far more conservative behaviour when it comes to the decision whether or not a sequence of words is considered a potential NE. Thus, whereas the Bi-LSTM$_{F:in}$ model recognizes even more than actually exist in the training set, the baseline classifier detects only 11,790 entities. However, since the Bi-LSTM-CRF networks reduce the number of identified NEs to approximately 13,500, the respective precision values improve by up to 7.39% compared to the baseline.

7.1.1 Quantitative Analysis on Entity Level

If the F_1 scores for the individual entity types are focused, the baseline and the best Bi-LSTM-CRF$_{F:in+W:out}$ model allow for almost identical performance rankings. In either case, the respective highest F_1 values are achieved for the *pers* and *loc* categories and the worst performance scores are noted for the *org* and *prod* types. Though, the CRF model is better at predicting entities of the *time* class than correctly identifying those of the type *amount*. The neural classifier, by contrast, shows an F_1 score of 72.93 for the *amount* and a value of 72.34 for the *time* category. Table 12 presents the precision, recall and F_1 values per entity type for the baseline as well as for the Bi-LSTM-CRF$_{F:in+W:out}$ model.

		Baseline model			**Best model**		
Entity type	**Frequency**	**P**	**R**	**F$_1$**	**P**	**R**	**F$_1$**
pers	3,638	75.61	70.97	73.21	84.55	84.72	84.64
func	3,133	69.65	43.66	53.68	76.48	66.84	71.33
loc	2,369	67.78	63.57	65.61	73.09	77.16	75.07
amount	1,716	63.44	53.79	58.22	75.00	70.98	72.93
time	1,675	70.43	59.28	64.38	73.43	71.28	72.34
org	1,412	63.85	44.41	52.38	67.61	52.48	59.09
prod	612	54.09	30.23	38.78	60.65	51.63	55.78

Table 12: Performance scores of the baseline and the best neural model for each entity type. Precision, recall and F_1 values are calculated based on the summed up TP, FP and FN counts over a category's subtypes.

As anticipated in Section 5.1.2.2, a decisive factor for the classifiers' relatively poor performance with respect to the *prod* category is to be located in the shortage of training examples. There are only 4,575 instances of this type in the training set, which comes up to a share of merely 4.43% of a total of 103,304 entities. Moreover, the *prod* category assembles nine subtypes, which implies that the individual subcategories are predicted based on an average of only 500 training examples.
Although performance scores are generally better for the *org* class, they lag considerably behind the ones for the remaining categories, at least in the case of the best NN. Here, the values for the two subtypes, i.e., the *org.ent* and the *org.adm* subtypes, are fairly unbalanced. While the neural classifier is able to properly identify 531 of the 803 *org.adm* entities (66.13% recall, 69.59 F_1), it recognizes only a third of the 609 *org.ent* NEs (34.4% recall, 42.77 F_1) in the test set. Difficulties regarding the correct recognition of the latter may be due, among other things, to a rather complex composition of the entities in question. Thus, whereas 45% of the 4,982

org.ent NEs in the training and test data are made up of three or more words, this is true for only 25% of the 6,240 *org.adm* entities.

In contrast to the merely 4,575 examples of the *prod* type, the training material holds 28,102 instances of the two *pers* subtypes. Therefore, more than a quarter of the example entities belong to either the *pers.ind* or the *pers.coll* subclass. As a consequence, when it comes to the identification and labelling of *pers* type entities, both the baseline (73.21 F_1) and the best neural model (84.64 F_1) achieve the by far highest performance scores.

For the *func*, *time*, *amount* and *loc* types, the Bi-LSTM-$CRF_{F:in+W:out}$ model reaches fairly even scores between 71.33 and 75.07 F_1. Concerning the *func* type, for which the neural classifier's performance is relatively low, rather uneven amounts of training instances for its two subclasses impede a potentially higher F_1 value. Thus, in contrast to the 12,932 *func.ind* examples, the training material contains merely 4,701 entities of the *func.coll* subtype. Accordingly, the neural classifier works better for the former (75.59 F_1) than for the latter (62.82 F_1) subcategory and, since the proportion of *func.ind* and *func.coll* NEs in the test set is somewhat more balanced, the overall performance for the *func* type is dragged down.

In the case of the *loc* category, more than half of the entities in the training data, and likewise in the test set, belong to either the *loc.adm.nat* or the *loc.adm.town* subtype. Regarding these two subcategories, the neural model works quite well (78.85 and 82.42 F_1, respectively), which offsets the sometimes poor performances for the other ten *loc* subtypes. Nevertheless, it should be noted that the Bi-LSTM-$CRF_{F:in+W:out}$ classifier reaches its overall best F_1 score (90.91) for the *loc.add.phys* subcategory. Although the training data holds less than 700 relevant examples, the 22 *loc.add.phys* entities in the test set are predicted correctly in the majority of cases. This is due to the fact that the expressions assignable to the subclass in question show fairly homogeneous token sequence patterns and mostly contain highly distinctive words, such as *rue* ('street'), *avenue* or *boulevard*.

Whereas the baseline model shows a performance range of 38.79 F_1 for the *prod* class to 73.21 F_1 for the *pers* type, the best NN is capable of balancing the respective scores somewhat more evenly. In the case of both classifiers, the reasons for the performance differences with respect to the seven entity types are diverse in nature. Eventually, the amount of training examples, the number of subtypes per category, the complexity of the entities and the existence of type-specific keywords or sequence patterns impact the performances for the different categories. An additional factor which has not been considered so far, but which likewise affects the sequence models performances, comes up to the OCR-conditioned spelling variation of the NEs. How the two classifiers compare against each other regarding to this latter issue is surveyed in the next section.

7.1.2 Performances for Misspelled Entities

For the assessment of the CRF and best neural model's performance concerning the identification and labelling of misspelled entities, it is taken advantage of the manually corrected OCR errors in the QOP data. As mentioned in Section 5.1.1.2, the annotators of the QOP corpus were instructed to record the rectified variants of those NEs which had not been recognized properly by the OCR tool. While there are different types of recognition errors (see Galibert et al., 2012, p. 3129), faulty OCR basically complicates the NER task in two ways. On the one hand, an entity may not be identified as such because one ore more of its elements contain miss-recognized characters and/or arbitrary noise. For instance, the first character of the word *fer* in Example (7.1) had been recognized as 'f' rather than as '1'. Accordingly, the sequence 'le fer avril', which was manually labelled as a *time.date.abs* type NE, exhibits one misspelled token. Therefore, a sequence classifier potentially fails to identify this sequence since it probably expects the presence of a token beginning with a number instead of a lower-case letter.

(7.1) L'association des industriels allemands veut *le fer avril* donner un grand éclat [...]

On the other hand, erroneous OCR may entail wrong word segmentation, which results in either separate tokens sticking together or individual tokens appearing as sequences of multiple words. In Example (7.2), the prepositions *de* ('from') and *en* ('in') are glued to the expressions *Cosènzà* and *Calabre*, respectively, since the intermediate blank space had not been recognized by the OCR tool. In such cases, the annotators of the QOP corpus were advised to tag the entire character span with the category of the contained NE. Thus, the sequence 'déCosènzà' is considered a *loc.adm.town* type entity and 'enCalabre' receives the label *loc.adm.reg*. Regarding the labelling of *loc* type entities, a sequence model probably learns that expressions which refer to locations exhibit an initial upper-case letter and, therefore, misses the two erroneous entities in (7.2).

(7.2) Il n'était pas Piémontais, mais bon Méridional, *déCosènzà enCalabre*.

(7.3) Ce journal croit qu'une sçission paraît inévitable dans le *parti liber pI* P

The opposite case of mistakenly split tokens is illustrated in Example (7.3). Here, the expression *parti libéral* ('Liberal Party'), which actually consists of two words, had been recognized as 'parti liber pI'. In addition to miss-recognized characters, the sequence is made up of three tokens, which means that a classifier has to identify three instead of two tokens in order to properly label the entire entity.

Since the provided QOP data files include the NE corrections in the form of XML-like attributes, plain text or IOB-formatted data may be generated with either corrected or noisy entities. Given these options, the latter was chosen for the creation of QOP_{IOB} corpus and information concerning NEs with an available OCR correction, i.e., NEs which contain noise or misspelled words, was retained. As mentioned in Section 5.1.2.2, a total of 982 erroneous entities can be located in the QOP_{IOB} test set.[31] Hence, for each of these entities may be determined whether or not it is labelled properly by the baseline and the Bi-LSTM-$\text{CRF}_{\text{F:in+W:out}}$ model. Table 13 list the corresponding raw counts of correctly ([+]) and wrongly ([−]) classified entities per NE type.

NE type	**Baseline** [+] **Best** [+]	**Baseline** [−] **Best** [+]	**Baseline** [−] **Best** [−]	**Baseline** [+] **Best** [−]
amount	32	22	34	5
func	49	57	84	7
loc	40	52	50	6
org	15	17	46	1
pers	124	70	58	7
prod	15	32	50	0
time	25	32	49	3
Total	300	282	371	29

Table 13: Number of misspelled entities which are correctly identified ([+]) and missed ([−]) by the baseline and best neural model.

Of the 982 entities which contain misspelled words, nearly a third is identified properly by both the CRF and the best neural model. In contrast, the two classifiers assign no or wrong labels to 371 erroneous NEs. However, the Bi-LSTM-$\text{CRF}_{\text{F:in+W:out}}$ network is able to correctly annotated 282 entities which are not identified by the baseline model. Furthermore, 29 NEs are detected by the CRF but not by the neural classifier. Hence, in absolute numbers, the LSTM-$\text{CRF}_{\text{F:in+W:out}}$ model retrieves 253 more relevant NEs than the CRF classifier, which comes up to an increase of almost 80%. In terms of recall, the value achieved by the neural model (59.40%) represents a considerable improvement over the one obtained by the baseline classifier (33.71%).

[31] The number of 982 faulty NEs comes up to 6.74% of the total of entities in the test data. Except for the *prod* type, which accounts for a 15.85 percent share, the portion of erroneous entities per category ranges between 5% and 7.5%. In this respect, the test set pretty much resembles the training data. In the case of the latter, there are corrections available for 17.49% of the *prod* type entities and for 6.5% to 9.5% of the NEs of each of the remaining classes. Overall, OCR rectified variants exist for 8,609 or 8.33% of the 103,304 entities in the training corpus.

In reference to the individual entity types, the highest proportions of NEs missed by the baseline as well as the neural model are observed for the *org* and the *prod* category. For these two categories, the relative figures of non-identified NEs amount to 58.23% and 51.55%, respectively. This is actually in line with what has been stated in Section 7.1.1 concerning the classifiers' performances for the *org* and *prod* type entities on the whole. Correspondingly, the misspelled NEs belonging to the *pers* class, for which the sequence models achieve the overall best results, exhibit the lowest proportion of misspelled entities which are missed by both classifiers. Figure 15 shows the relative numbers of identified and non-recognized entities for each of the seven categories.

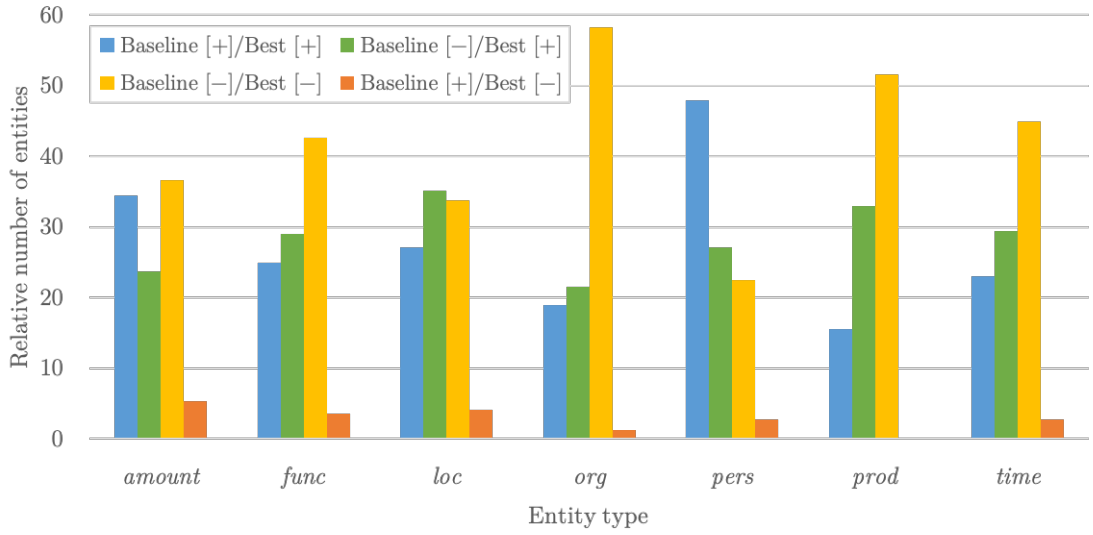


Figure 15: Proportions of misspelled entities which are correctly identified ([+]) and missed ([−]) entities per entity type.

The average CER of the 300 NEs which are correctly identified by the baseline as well as the neural model amounts to 16.92%. By contrast, the CERs for the entities which are properly labelled by only the Bi-LSTM-CRF$_{\text{F:in+W:out}}$ network average out at 20.54%. This allows for the conclusion that the neural model, if compared to the baseline, performs better for erroneous entities with higher proportions of miss-recognized characters and/or noise. However, an average error rate of 22.76% for the NEs which are not recognized by any of the two classifiers suggests that the neural classifier also has its limitations. These limitations, as well as the concrete situations in which the Bi-LSTM-CRF$_{\text{F:in+W:out}}$ model manages to outperform the baseline, are focused in the next section.

7.1.2.1 Misspelled Names

The *amount* and *time* types actually cover numerical and temporal expressions, respectively, and the *func* category is intended to capture academic and aristocratic titles as well as employment titles associated with individuals. While the entities belonging to these classes do not necessarily contain proper NEs, the *loc*, *org*, *pers* and *prod* type entities conform to NEs in a narrow sense. Thus, the latter entities effectively refer to concrete abstract or physical objects in the real world and are mostly denoted with unambiguous expressions. As illustrated in Table 14, the baseline model is able to correctly identify such entities if they contain additional distinctive words. For instance, the token *M.* in (1) and (2), which corresponds to the abbreviation of *monsieur* ('mister'), frequently appears in *pers* type entities. This allows for a straightforward identification of the NEs in question, even if the actual person name is affected by OCR-conditioned errors.

	Token		**IOB**		
	OCR output	**corrected**	**Gold**	**Baseline**	**Best**
(1)	M.	M.	B-pers.ind	B-pers.ind	B-pers.ind
	Vign&ux	Vignaux	I-pers.ind	I-pers.ind	I-pers.ind
(2)	M.	M.	B-pers.ind	B-pers.ind	B-pers.ind
	Dulî'our	Duffour	I-pers.ind	I-pers.ind	I-pers.ind

Table 14: Instances of misspelled *pers* type entities which are correctly identified by the baseline as well as the best neural model.

Generally, if there are no distinctive tokens, the CRF classifier shows weaknesses especially in the case of erroneously introduced punctuation marks and special characters. Likewise, names starting with miss-recognized lower-case letters, which is rather atypical for *loc*, *pers* and *prod* type entities, pose problems for the baseline model. By contrast, the Bi-LSTM-CRF$_{\text{F:in+W:out}}$ network proves to be quite less sensitive to such kinds of OCR errors. Examples (1) to (4) in Table 15 show instances of misspelled entities which are identified by the neural model but not the baseline classifier. Here, the fourth example represents the only case for which the CRF model achieves at least a partial match. Since the shape of the correctly spelled second token (*Chêne*) conforms to the characteristics of a person name, i.e., it is composed of an initial upper-case letter followed by lower-case letters, the classifier is capable of labelling it as a *pers.ind* type entity. Though, the erroneous first token (*o°an*) is made up of lower-case letters only, and an additional special character, which impedes the CRF classifier to identify it as being of the same entity type as the following token.

	Token		IOB		
	OCR output	**corrected**	**Gold**	**Baseline**	**Best**
(1)	T«i*louse	Toulouse	B-loc.adm.town	O	B-loc.adm.town
(2)	Caa.Qrs	Cahors	B-loc.adm.town	O	B-loc.adm.town
(3)	RooxEb	ROUXEL	B-pers.ind	O	B-pers.ind
	t*lCRAJ	LECRAI	I-pers.ind	O	I-pers.ind
(4)	o°an	Jean	B-pers.ind	O	B-pers.ind
	Chêne	Chêne	I-pers.ind	B-pers.ind	I-pers.ind

Table 15: Instances of misspelled *loc* and *pers* type entities which are identified by the best neural model but not the baseline classifier.

Indeed, the cases in which only the Bi-LSTM-CRF$_{\text{F:in+W:out}}$ network assigns proper entity labels include some fairly astonishing examples. For instance, the test set contains three misspelled variants of the expression *Temps* which bear little or no resemblance to the corrected version. However, examples (1) to (3) in Table 16 show that the neural classifier correctly labels them as *prod.media* (media product) type entities. Whereas these highly distorted entities are understandably missed by the baseline model, the erroneous expression *UEçfijptian Gazette* in example (4) contains at least one properly spelled element. In addition, the token *Gazette* may be considered distinctive for the *prod.media* category since it appears in more than 70 relevant entities in the training set. Nevertheless, the CRF model assigns this instance a wrong *pers.ind* label.

	Token		IOB		
	OCR output	**corrected**	**Gold**	**Baseline**	**Best**
(1)	©cmp0	Temps	B-prod.media	O	B-prod.media
(2)	f&tvxps	Temps	B-prod.media	O	B-prod.media
(3)	f&tïtipB	Temps	B-prod.media	O	B-prod.media
(4)	UEçfijptian	Egiptian	B-prod.media	B-pers.ind	B-prod.media
	Gazette	Gazette	I-prod.media	I-pers.ind	I-prod.media

Table 16: Instances of misspelled *prod* type entities which are identified by the best neural model but not the baseline classifier.

The NEs which are correctly identified by the baseline but not the neural model likewise comprise a few rather surprising examples. Regarding the first instance in Table 17, the expression *Jeanaa d'Arc* exhibits a comparatively low portion of miss-recognized characters and is correctly identified by the CRF classifier as a *pers.ind*

type entity. Furthermore, the training data holds roughly 20 occurrences of correct or misspelled variants of *Jeanne d'Arc*, which are mostly labelled as *pers.ind* or *prod.art* (work of art) but never as *loc.adm.town* type entities. Notwithstanding this, the Bi-LSTM-CRF$_{\text{F:in+W:out}}$ network assigns the expression the incorrect *loc.adm.town* label. Similarly, the correct variant of the expression in the second example figures in the training set as a *loc.adm.town* entity. Though, the neural model mistakenly labels *Sables-d'Olonnc* as a *pers.coll* type NE.

	Token		**IOB**		
	OCR output	**corrected**	**Gold**	**Baseline**	**Best**
(1)	Jeanaa	Jeanne	B-pers.ind	B-pers.ind	B-loc.adm.town
	d'	d'	I-pers.ind	I-pers.ind	I-loc.adm.town
	Arc	Arc	I-pers.ind	I-pers.ind	I-loc.adm.town
(2)	Sables-d'Olonnc	Sables-d'Olonne	B-loc.adm.town	B-loc.adm.town	B-pers.coll

Table 17: Instances of misspelled *pers* and *loc* type entities which are identified by the baseline model but not the best neural classifier.

Eventually, the last two examples show that the predictions of the Bi-LSTM-CRF classifier are sometimes difficult to understand. Thus, it is not clear at all why the neural model seemingly ignores that there are no variants of *Jeanne d'Arc* in the training data which are labelled as a *loc.adm.town* entity.

7.1.2.2 Digits and Distinctive Words

As mentioned above, the *amount*, *time* and *func* types are intended to cover numerical and temporal expressions and different kinds of titles, respectively. For instance, the entities belonging to the *time* category typically contain digits and/or distinctive words, such as units of time or month names. As exemplified by means of the erroneous expressions *l»r mai* and *8 hwires* in Table 18, the baseline classifier runs into problems if either a numeral or a type-specific word exhibits spelling errors. By contrast, despite of the missing digit in (1) and the misspelled time unit in (2), the Bi-LSTM-CRF$_{\text{F:in+W:out}}$ network is capable of labelling both expressions correctly. The same applies to the labelling of the misspelled *amount* and *func* type NEs in Table 19. Here, the first and second examples illustrate instances of *amount* entities which contain numeral elements affected by erroneous OCR. However, while the CRF model completely misses the entity in (1), it produces at least a partial match for the expression in (2). In the case of examples (3) and (4), the two tokens *docteur* and *ministre*, which can be considered distinctive for the *func* category, are spelled

incorrectly and, consequently, the baseline classifier fails to identify the respective NEs.

	Token		**IOB**		
	OCR output	**corrected**	**Gold**	**Baseline**	**Best**
(1)	l»r	1er	B-time.date.abs	I-org.ent	B-time.date.abs
	mai	mai	I-time.date.abs	I-org.ent	I-time.date.abs
(2)	8	8	B-time.hour.abs	B-amount	B-time.hour.abs
	hwires	heures	I-time.hour.abs	I-amount	I-time.hour.abs

Table 18: Instances of misspelled *time* type entities which are identified by the best neural model but not the baseline classifier.

	Token		**IOB**		
	OCR output	**corrected**	**Gold**	**Baseline**	**Best**
(1)	A	4	B-amount	O	B-amount
	millions	millions	I-amount	B-amount	I-amount
	de	de	I-amount	I-amount	I-amount
	francs	francs	I-amount	I-amount	I-amount
(2)	i	7	B-amount	O	B-amount
	shillings	shillings	I-amount	O	I-amount
(3)	dScïeur	docteur	B-func.ind	O	B-func.ind
(4)	joiinistro	ministre	B-func.ind	O	B-func.ind
	de	de	I-func.ind	O	I-func.ind
	la	la	I-func.ind	O	I-func.ind
	marine	marine	I-func.ind	O	I-func.ind

Table 19: Instances of misspelled *amount* and *func* type entities which are identified by the best neural model but not the baseline classifier.

Whereas the presence of a properly spelled type-specific token may support the CRF model in the correct identification and labelling of an NE, distinctive words which contain OCR-conditioned errors rather prove to be an inhibiting factor. Likewise, the baseline model relies on numerals for the recognition of *amount* and *time* type entities. Accordingly, if these numerals exhibit spelling errors, i.e., if they appear converted into letters or other non-numeric characters, the CRF classifier shows considerable weaknesses with respect to the labelling of the entities in question. On the contrary, the Bi-LSTM-CRF$_{\text{F:in+W:out}}$ network depends less on the correct spelling of occurring distinctive words and digits and, therefore, succeeds in identifying entities which are not or wrongly labelled by the baseline model.

7.1.2.3 Word Segmentation Errors

It has already been indicated that erroneous OCR may result in neighbouring tokens being glued together and/or mistakenly split words. This latter phenomenon is further intensified by the tokenization preprocessing of the data. For instance, the tokenizer splits the expression *M. "Wagon* in Example (7.4) into three units since it recognizes the erroneously introduced double quote as a punctuation mark and, therefore, as a separate token.

(7.4) [...] on cite d'avance celle de *M. "Wagon*, pharmacien, chef du parti radical.

As a consequence of word segmentation errors, a sequence classifier has to identify and label an either larger or smaller number of tokens than there actually are in the correct word sequence. As shown in Table 20, both the baseline and the neural model seem to struggle with instances of non-entity tokens sticking to an NE token. Thus, in the case of all three examples, neither the Bi-LSTM-CRF$_{\text{F:in+W:out}}$ network nor the CRF classifier recognizes the erroneous expressions as NEs.

	Token		**IOB**		
	OCR output	**corrected**	**Gold**	**Baseline**	**Best**
(1)	déCosènzà	de Cosenza	B-loc.adm.town	O	O
(2)	enCalabre	en Calabre	B-loc.adm.reg	O	O
(3)	AGenappe	A Genappe	B-loc.adm.town	O	O

Table 20: Instances of preposition glued to *loc* type entities which are identified neither by the baseline nor the best neural model.

On the other hand, the neural classifier is able to correctly identify entities with wrongly split tokens which are not or only partly labelled by the baseline model. In addition to the faulty word segmentation, the expressions in the examples in Table 21 exhibit relatively high CERs of 30% to 50%. Actually, the expressions in (2) and (3) contain hyphenated tokens, which generally exhibit increased proportions of misspelled characters and noise. Thus, given a notable amount of layout-conditioned hyphenations, the neural model's capability of handling such instances proves especially beneficial with respect to historical newspaper data.

	Token		IOB		
	OCR output	**corrected**	**Gold**	**Baseline**	**Best**
(1)	1	1200	B-amount	O	B-amount
	ZOO		I-amount	O	I-amount
	lettres	lettres	I-amount	O	I-amount
(2)	éminentaex«	éminents	B-func.coll	O	B-func.coll
	!		I-func.coll	O	I-func.coll
	perts	experts	I-func.coll	O	I-func.coll
(3)	Marie-	Marie-Made-leine	B-pers.ind	B-pers.ind	B-pers.ind
	!		I-pers.ind	O	I-pers.ind
	|#4e«		I-pers.ind	O	I-pers.ind
	leine		I-pers.ind	O	I-pers.ind

Table 21: Instances of entities containing wrongly segmented tokens which are identified by the best neural model but not the baseline classifier.

7.2 Model Applicability to New Data

In terms of an additional experiment, the CRF and the best neural model were applied to newspaper texts which are not contained in the QOP corpus. To this, six manually annotated documents were selected from the digitized documents generated in the context of the *impresso* project. The chosen documents, which contain approximately 5,200 tokens, include articles from late 19th century issues[32] of the *Journal de Genève*, the *Gazette de Lausanne* and *L'Express*.

Actually, the results from the evaluation on the additional test data have to be treated with caution. Basically, the annotation scheme used for the *impresso* project corresponds to the one employed for the annotation of the QOP corpus. However, there are some considerable differences with respect to the set of entity types and subtypes as well as the applied annotation guidelines. On the one hand, the *impresso* data is labelled according to a reduced *Quaero* annotation scheme. On the other hand, a central directive of the *impresso* guidelines restricts the annotations to those entities which effectively contain a proper name. For instance, the expressions *anciens monarchistes* ('former monarchists') and *impérialistes découragés* ('disheartened imperialists') in Example (7.5) are annotated in the QOP corpus as *pers.coll* type entities. By contrast, these examples would not be labelled in the *impresso* data since they do not include proper names.

[32] All chosen documents were issued in 1888.

(7.5) Des comités se constituent dans les départements pour recommander cette méthode aux *anciens monarchistes* et aux *impérialistes découragés*.

Given the limited compatibility between the *Quaero* and the *impresso* annotation conventions, the experimental evaluation of the CRF and neural classifiers on the additional test data is restricted to the performances for the *loc* and *pers* type entities. Since the subcategories of these two categories are fundamentally the same in the case of both annotation guidelines, the only aspect which has to be taken into account relates to the different handling of entities which do not contain proper names. The performance scores of the baseline and the best neural model for the *loc* and *pers* entities in the experimental test set are shown in Table 22.

		Baseline model			**Best model**		
Entity type	**Frequency**	**P**	**R**	**F_1**	**P**	**R**	**F_1**
loc	132	56.67	25.76	35.42	64.52	45.45	53.33
pers	200	50.54	46.27	48.31	68.00	69.00	68.50

Table 22: Performance scores of the baseline and the best neural model for *loc* and *pers* type entities in the additional *impresso* data. Precision, recall and F_1 values are calculated based on the summed up TP, FP and FN counts over a category's subtypes.

In the case of both entity types, the Bi-LSTM-$CRF_{F:in+W:out}$ network clearly outperforms the CRF model by roughly 20 points F_1. In comparison to the performance scores for the *loc* and *pers* entities in the QOP_{IOB} test data, the neural classifier shows decreases in F_1 of 21.74 and 16.14, respectively. The low recall (45.45%) for the *loc* category is mainly due to almost 30 non-identified *loc.add.phys* NEs which figure in one of the experimental test documents. Moreover, the neural model shows weaknesses with respect to the differentiation between mentions of Swiss cantons (*loc.adm.reg*) and towns (*loc.adm.town*). This is probably due to the fact that the newspaper documents in the classifiers' training corpus were issued in France. By contrast, the articles included in the experimental data set were published in Switzerland.

Regarding the performance scores for the *pers* class, the labelling of entities which do not contain proper names actually represents an issue. Thus, there are at least 14 wrong identifications of *pers.coll* entities which may be attributed to this discrepancy between the two annotation guidelines. On the other hand, the neural model is able to correctly label 137 *pers.ind* type NEs, even though the *impresso* data exhibits a slightly different tokenization. For instance, abbreviation associated periods are treated as a separate token, which effects that abbreviated forenames or titles in *pers* entities appear as two tokens. While this circumstance frequently

irritates the CRF model, it does not pose a problem to the neural classifier.
Eventually, the divergences between the original and the modified *Quaero* annotation scheme and guidelines prevent a comprehensive assessment of the Bi-LSTM-$CRF_{F:in+W:out}$ network's performance on the *impresso* data. However, a fairly acceptable F_1 value for at least the *pers* type NEs suggest that the model trained on the QOP_{IOB} corpus could be successfully applied to the digitized articles of other historical newspapers.

8 Conclusion

In this thesis, a neural approach for NER in digitized historical texts has been described. The data on which the NER task was performed corresponds to a French language newspaper corpus with OCRed and manually annotated documents from the end of the 19th century. Due to erroneous OCR, the used data exhibits a notable amount of miss-recognized characters and noise, which complicates the identification and classification of the contained NEs.

Considered as a sequence labelling problem, NER in the digitized newspaper documents was tackled by means of Bi-LSTM-based classifiers which are robust in the face of OCR and historically conditioned spelling variation. To this, the employed neural sequence models rely on a particular type of word embeddings which model words as strings of contextualized characters. These contextualized string embeddings are generated with the help of neural character-level LMs which are trained to predict the next character in a sequence given the previous characters.

In the context of this thesis, neural character-level LMs were trained on in-domain, out-of-domain and mixed-domain data in order to assess which of the three LM types contributes most to NER in OCRed newspaper texts. In this respect, it has been found that contextual string embeddings created with the LMs trained on noisy historical in-domain data most favourably affect the performances of the neural sequence classifiers and applying embeddings produced by means of the LMs trained on contemporary *Wikipedia* articles results in the worst performances scores.

On the other hand, it has been shown that the neural sequence models relying on in-domain contextual string embeddings thoroughly outperform the baseline CRF classifier which makes use of a series of hand-crafted spelling features. The highest overall performance gains over the baseline model amount to approximately 12 points F_1. Likewise, the neural classifiers reaches better performance scores for all of the considered NE categories. However, the classes for which the compared models achieve the respective worst and best performance scores are identical for the baseline and the neural classifiers.

In terms of an additional experiment, the baseline CRF and the Bi-LSTM-based sequence models were applied to historical newspaper data from a document collection different from the one used for the training of the compared classifiers. To

this, only the results for the *loc* and *pers* type entities were taken into account since the experimental test set notably differs from the original data with regard to the employed annotation scheme and guidelines. While the CRF model achieves performance scores of less than 50 F_1 for the considered NE types, the neural classifier performs acceptably for at least the entities of the *pers* category (almost 70 F_1). Unfortunately, the lack of corpora annotated according to the tagging scheme and directives of the initially used historical newspaper data prevents a more comprehensive evaluation of the applicability of the neural models trained for this work to texts from other newspapers.

Eventually, the presented results show that neural sequence classifiers which rely on contextual string embeddings clearly outperform classical feature-based CRF models in the task of NER in digitized historical texts. The key part of the success of the chosen approach is located in the used embedding type's capability of modelling words as sequences of contextualized characters. This form of representing words greatly favours the identification and labelling of misspelled expressions, which are inherent to the data focused in the context of this thesis.

References

A. Akbik, D. Blythe, and R. Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING 2018)*, pages 1638–1649, Santa Fe, New Mexico, USA, 2018. Association for Computational Linguistics.

A. Akhundov, D. Trautmann, and G. Groh. Sequence labeling: A practical approach. *Computing Research Repository (CoRR)*, arXiv:1808.03926 [cs.CL], August 2018.

D. Benikova, S. M. Yimam, P. Santhanam, and C. Biemann. Germaner: Free open german named entity recognition tool. In *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. German Society for Computational Linguistics and Language Technolog e.V., 2015.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

M. Bollmann. A large-scale comparison of historical text normalization systems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3885–3898, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

J. P. C. Chiu and E. Nichols. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics (TACL)*, 4:357–370, 2016.

G. Crane and A. Jones. The challenge of virginia banks: An evaluation of named entity analysis in a 19th-century newspaper collection. In *Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL '06)*, pages 31–40, Chapel Hill, NC, USA, June 2006. Association for Computing Machinery.

M. Dinarelli and S. Rosset. Tree-structured named entity recognition on ocr data: Analysis, processing and results. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, pages 1266–1272, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).

J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL '05)*, ACL '05, pages 363–370, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

O. Galibert, L. Quintard, S. Rosset, P. Zweigenbaum, C. Nédellec, S. Aubin, L. Gillard, J.-P. Raysz, D. Pois, X. Tannier, L. Deléger, and D. Laurent. Named and specific entity detection in varied data: The quæro named entity baseline evaluation. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC 2010)*, pages 3453–3458, Valletta, Malta, May 2010. European Languages Resources Association (ELRA).

O. Galibert, S. Rosset, C. Grouin, P. Zweigenbaum, and L. Quintard. Extended named entity annotation on ocred documents: From corpus constitution to evaluation campaign. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, Istanbul, Turkey, May 2012. European Languages Resources Association (ELRA).

Y. Goldberg. *Synthesis Lectures on Human Language Technologies*, volume 10 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers, April 2017.

J. Graën, M. Bertamini, and M. Volk. Cutter – a universal multilingual tokenizer. In *Swiss Text Analytics Conference*, number 2226 in CEUR Workshop Proceedings, pages 75–81. CEUR-WS, June 2018.

C. Grouin, S. Rosset, P. Zweigenbaum, K. Fort, O. Galibert, and L. Quintard. Proposal for an extension of traditional named entities: From guidelines to evaluation, an overview. In *Proceedings of the 5th Linguistic Annotation Workshop (LAW V '11)*, pages 92–100, Portland, Oregon, USA, 2011. Association for Computational Linguistics.

C. Grover, S. Givon, R. Tobin, and J. Ball. Named entity recognition for digitised historical texts. In *Proceedings of the Sixth International Conference on*

Language Resources and Evaluation (LREC 2008), pages 1343–1346, Marrakech, Morocco, May 2008. European Language Resources Association (ELRA).

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

R. Holley. Crowdsourcing: How and why should libraries do it? *D-Lib Magazine*, 16, March 2010.

Z. Huang, W. Xu, and K. Yu. Bidirectional LSTM-CRF models for sequence tagging. *Computing Research Repository (CoRR)*, arXiv:1508.01991 [cs.CL], August 2015.

D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition.* Prentice Hall, Upper Saddle River, NJ, USA, 3rd (draft) edition, 2018.

J. D. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01)*, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. 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.

T. Lavergne, O. Cappé, and F. Yvon. Practical very large scale CRFs. In J. Hajič, S. Carberry, and S. C. andJoakim Nivre, editors, *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 504–513, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

X. Ma and E. 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 (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany, August 2016. Association for Computational Linguistics.

S. Mac Kim and S. Cassidy. Finding names in trove: Named entity recognition for australian historical newspapers. In *Proceedings of the Australasian Language Technology Association Workshop 2015*, pages 57–65, Parramatta, Australia, December 2015.

A. McCallum. Mallet: A machine learning for language toolkit. 2002.

A. McCallum, D. Freitag, and F. C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00)*, pages 591–598, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS '13)*, volume 2, pages 3111–3119, Lake Tahoe, Nevada, 2013. Curran Associates Inc.

C. Neudecker. An open corpus for named entity recognition in historic newspapers. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 4348–4352, Paris, France, May 2016. European Language Resources Association (ELRA).

M. Piotrowski. *Natural Language Processing for Historical Texts*, volume 17 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers, September 2012.

N. Reimers and I. 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.

M. Riedl and S. 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, 2018. Association for Computational Linguistics.

S. Rosset, C. Grouin, and P. Zweigenbaum. Entités nommées structurées : guide d'annotation quaero. Notes et Documents LIMSI 2011-04, Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur, September 2011.

S. Rosset, C. Grouin, K. Fort, O. Galibert, J. Kahn, and P. Zweigenbaum. Structured named entities in two distinct press corpora: Contemporary broadcast news and old newspapers. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 40–48, Jeju, Republic of Korea, July 2012. Association for Computational Linguistics.

E. F. Tjong Kim Sang and F. 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 (CoNLL-2003)*, pages 142–147, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

S. Yan. Understanding LSTM and its diagrams. https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714, March 2016.

Lebenslauf

Persönliche Angaben

Stefan Bircher
Antonigasse 12
5620 Bremgarten
stefan.bircher@uzh.ch

Schulbildung

2008-2016	Bachelor-Studium Geschichte, Spanische Sprachwissenschaft und Computerlinguistik an der Universität Zürich
2016-2019	Master-Studium Multilinguale Textanalyse (MLTA) an der Universität Zürich

Berufliche und nebenberufliche Tätigkeiten

2010–2019	Mitarbeiter Informatikdienste ETH Zürich
2016	Tutorat Arbeit im Spätmittelalter an der Universität Zürich

A Tables

QOP_{IOB} test set		**QOP_{IOB} dev set**	
0212650_4.norm.ne	0232778_1.norm.ne	0212650_1.norm.ne	0232780_2.norm.ne
0212651_4.norm.ne	0232778_2.norm.ne	0212650_2.norm.ne	0232780_3.norm.ne
0212652_1.norm.ne	0232779_1.norm.ne	0212650_3.norm.ne	0232781_1.norm.ne
0212652_2.norm.ne	0232779_2.norm.ne	0212651_1.norm.ne	0232781_2.norm.ne
0212652_3.norm.ne	0232779_3.norm.ne	0212651_2.norm.ne	0232781_3.norm.ne
0212653_1.norm.ne	0232780_4.norm.ne	0212651_3.norm.ne	0232782_4.norm.ne
0212653_2.norm.ne	0232781_4.norm.ne	0212652_4.norm.ne	0232783_4.norm.ne
0212653_3.norm.ne	0232782_1.norm.ne	0212653_4.norm.ne	0232786_1.norm.ne
0232774_1.norm.ne	0232782_2.norm.ne	0232774_4.norm.ne	0232786_2.norm.ne
0232774_2.norm.ne	0232782_3.norm.ne	0232775_4.norm.ne	0232786_3.norm.ne
0232774_3.norm.ne	0232783_1.norm.ne	0232776_2.norm.ne	0232787_1.norm.ne
0232775_1.norm.ne	0232783_2.norm.ne	0232776_3.norm.ne	0232787_2.norm.ne
0232775_2.norm.ne	0232783_3.norm.ne	0232777_1.norm.ne	0232787_3.norm.ne
0232775_3.norm.ne	0232786_4.norm.ne	0232777_2.norm.ne	0281337_2.norm.ne
0232776_1.norm.ne	0232787_4.norm.ne	0232777_3.norm.ne	
0232776_4.norm.ne	0281337_1.norm.ne	0232779_4.norm.ne	
0232777_4.norm.ne		0232780_1.norm.ne	

Table 23: QOP test files used to generate QOP_{IOB} test and development sets

	F1 score						
Model	**#1**	**#2**	**#3**	**#4**	**#5**	**Mean**	**Stdev**
Bi-LSTM$_{\text{F:out}}$	68.85	68.26	68.67	68.48	68.69	68.59	0.20
Bi-LSTM$_{\text{F:out+W:out}}$	68.70	68.85	68.94	68.68	68.77	68.79	0.11
Bi-LSTM$_{\text{F:out+W:mix}}$	68.90	69.04	69.24	68.89	69.01	69.02	0.14
Bi-LSTM-CRF$_{\text{F:out}}$	72.67	72.30	72.58	72.65	72.55	72.55	0.15
Bi-LSTM-CRF$_{\text{F:out+W:out}}$	73.20	73.25	72.54	72.64	72.90	72.91	0.32
Bi-LSTM-CRF$_{\text{F:out+W:mix}}$	73.11	73.24	73.01	73.22	72.91	73.10	0.14
Bi-LSTM$_{\text{F:in}}$	68.83	69.02	69.11	69.36	68.91	69.05	0.21
Bi-LSTM$_{\text{F:in+W:out}}$	69.41	69.69	69.30	69.55	69.68	69.53	0.17
Bi-LSTM$_{\text{F:in+W:mix}}$	69.65	69.81	69.65	69.48	69.63	69.64	0.12
Bi-LSTM-CRF$_{\text{F:in}}$	72.67	72.75	72.70	73.06	72.78	72.79	0.16
Bi-LSTM-CRF$_{\text{F:in+W:out}}$	74.00	73.54	73.40	73.61	73.60	73.63	0.22
Bi-LSTM-CRF$_{\text{F:in+W:mix}}$	73.49	74.17	73.44	73.64	73.11	73.57	0.39
Bi-LSTM$_{\text{F:mix}}$	69.01	68.83	69.23	69.43	69.01	69.10	0.23
Bi-LSTM$_{\text{F:mix+W:out}}$	69.13	69.60	69.88	69.42	69.03	69.41	0.35
Bi-LSTM$_{\text{F:mix+W:mix}}$	69.32	69.47	69.50	68.95	69.64	69.38	0.26
Bi-LSTM-CRF$_{\text{F:mix}}$	72.85	73.32	73.34	73.22	73.18	73.18	0.20
Bi-LSTM-CRF$_{\text{F:mix+W:out}}$	73.36	73.63	73.08	73.36	73.54	73.39	0.21
Bi-LSTM-CRF$_{\text{F:mix+W:mix}}$	73.21	73.62	73.33	73.40	73.37	73.39	0.15
CRF	61.97	61.81	62.13	61.87	62.02	61.96	0.13

Table 24: F_1 scores for all neural models which were trained in the context of the configuration experiment. The subscripts *F:out*, *F:in* and *F:mix* refer to out-of-domain, in-domain and mixed-domain *Flair* embeddings, respectively. *W:out* relates to out-of-domain *fastText* embeddings and *W:mix* to mixed-domain *fastText* embeddings.

B Scripts

This appendix contains a list of scripts that were used in the context of this thesis. They can be found at `https://github.com/stbirc/quaero_ner/tree/master/lib`.

- Scripts used for evaluation:

  F.1 `conlleval.pl`
  Used to estimate the performance scores of the trained sequence classifiers.

- Scripts which were written and used in the context of this thesis:

  S.1 `quaero2iob.py`
  Used to convert the original QOP corpus into the IOB formatted QOP_{IOB} corpus.

  S.2 `train_cm.py`
  Used to train neural character-level LMs.

  S.3 `train_tagger.py`
  Used to train the neural sequence models which rely on *Flair* embeddings.

C Miscellaneous

C.1 *Wapiti* Feature Template

```
B
U

# the current word (case-sensitive)
U:word=%x[0,0]

# character prefixes and suffixes
U:suf-3 0=%m[0,0,"...$"]
U:pre-3 0=%m[0,0,"^..."]

# character shape features
*:starts-upper X=%t[0,0,"^\u"]
*:starts-lower X=%t[0,0,"^\l"]
*:all-upper X=%t[0,0,"^\u*$"]
*:mixed-case X=%t[0,0,"\l\u"]
*:ends-punct X=%t[0,0,"\p$"]
*:internal-punct X=%t[0,0,"\w\p\w"]
*:no-alpha X:%t[0,0,"^\A+$"]
*:start-end-number X:%t[0,0,"^\d"]/%t[0,0,"\d$"]
```

Universität Zürich[UZH]

Philosophische Fakultät
Studiendekanat

Universität Zürich
Philosophische Fakultät
Studiendekanat
Rämistrasse 69
CH-8001 Zürich
www.phil.uzh.ch

Selbstständigkeitserklärung

Hiermit erkläre ich, dass die Masterarbeit von mir selbst ohne unerlaubte Beihilfe verfasst worden ist und ich die Grundsätze wissenschaftlicher Redlichkeit einhalte (vgl. dazu: http://www.uzh.ch/de/studies/teaching/plagiate.html).

Zürich, 15.06.2019 .....

Ort und Datum Unterschrift