BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

NAMED ENTITY RECOGNITION EXPLOITING SUBWORD INFORMATION

ROZPOZNÁVANIE NÁZVOSLOVNÝCH ENTÍT VYUŽÍVAJÚCI PODSLOVNÉ INFORMÁCIE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR PATRIK DOBROVODSKÝ
AUTOR PRÁCE

SUPERVISOR SANTOSH KESIRAJU, Ph.D.
VEDOUCÍ PRÁCE

BRNO 2021

Department of Computer Graphics and Multimedia (DCGM) Academic year 2021/2022

Bachelor's Thesis Specification

24847

Student:	**Dobrovodský Patrik**
Programme:	Information Technology
Title:	**Named Entity Recognition Exploiting Sub Word Information**
Category:	Speech and Natural Language Processing

Assignment:

1. Study and understand current baseline systems for Named entity recognition (NER) that use LSTMs, CNNs, conditional random fields.
2. Select standard NER datasets from at least 2 Indo-European languages, apart from standard English NER dataset (CoNLL 2003).
3. Create baseline and state-of-the-art systems from the literature, for the selected datasets.
4. Create a framework for implementing NER models and various exploring architectures for incorporating sub-word information into the NER models. Suggested libraries: PyTorch, tensorflow.
5. Conduct experiments, compare the proposed models with state-of-the-art systems and analyze the results.

Recommended literature:

- https://aclanthology.org/Q16-1026.pdf
- List of freely available NER datasets: https://huggingface.co/datasets?sort=downloads&search=NER
- https://www.isca-speech.org/archive/interspeech_2019/abujabal19_interspeech.html

Requirements for the first semester:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Kesiraju Santosh**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2021
Submission deadline:	May 11, 2022
Approval date:	November 1, 2021

Abstract

The aim of this thesis is the creation of a Named Entity Recognition system based on an older state-of-the-art model and studying how subword information can improve the recognition of out-of-vocabulary words. This proposed system besides English has to support two additional Indo-European languages: German and Hungarian. This work features a named entity tagger based on deep learning using pretrained and custom-trained word embeddings, sparse features, and character embeddings extracted by a Convolutional Neural Network. All these features are then processed by sequence-based (bidirectional Long Short-Term Memory) and feature-based (Conditional Random Field) approaches with the goal of achieving a F1-score similar to the work it is based on, and to compare how far present time state-of-the-art systems have evolved. The result is a system that achieves a 90.98% F1-score on the CoNLL 2003 English test dataset using pretrained word embeddings, not far behind the original work's 91.26%. For the other two languages, the model scores 89.34% on the WikiAnn German test dataset and 93.04% on the WikiAnn Hungarian test dataset with the usage of custom-trained embeddings.

Abstrakt

Cieľom tejto bakalárskej práce je zhotovenie systému rozpoznania názvoslovnej entity zhotovenej na základe modelu, ktorý bol nedávno považovaný za jeden z najmodernejších a popri tom skúma aký vplyv majú podslovné informácie na nahradenie slov mimo slovnej zásoby. Vytvorený systém vedľa anglického jazyka podporuje aj dva Indo-Európske jazyky konkrétne nemčinu a maďarčinu. Bakalárska práca predstavuje systém využívajúci hlboké učenie pre rozpoznávanie názvoslovných entít, ktorý používa predtrénované a samotrénované slovné vnorenia, zriedkavé vnorenia a charakterové vnorenia vyzdvihnuté konvolučnou neurónovou sieťou. Tieto vnorenia najprv spracujeme sekvenčnou (dlhodobá-krátkodobá pamäť) a potom charakteristickou (podmienené náhodné pole) metódou. Cieľom je dosiahnuť podobnú F1-mieru akú má inšpiračný model s možnosťou porovnania s ostatnými modernými systémami. Výsledkom našej práce je systém, ktorý na anglickej testovacej sade CoNLL 2003 dosiahol 90.98%-né F1-mieru používajúci predtrénované vnorenia a približuje sa k inšpiračnej práci s hodnotou 91.26%. V prípade ďalších jazykov používajúcich samotrénované slovné vnorenia dosiahol systém na testovacej sade WikiAnn pre nemčinu 89.34%-nú a pre maďarčinu 93.04%-nú F1-mieru.

Keywords

Natural Language Processing, Named Entity Recognition, neural networks, Convolutional Neural Network, Conditional Random Fields, Long Short-Term Memory, subword information

Kľúčové slová

spracovanie prirodzeného jazyka, rozpoznávanie názvoslovných entít, neurónové siete, konvolučná neurónová sieť, podmienené náhodné pole, krátkodobá-dlhodobá pamäť, podslovné informácie

Reference

DOBROVODSKÝ, Patrik. *Named Entity Recognition Exploiting Subword Information.* Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Santosh Kesiraju, Ph.D.

Rozšírený abstrakt

V rýchlo sa rozvíjajúcom svete kde značné percento dát je prítomné v textovej forme je čoraz dôležitejšie, aby sme názvoslovné entity vedeli rýchlo a presne vyfiltrovať. Na dosiahnutie tohoto cieľa použili umelé neurónové siete, čím vytvorili nové odvetvie spracovania prirodzeného jazyka – spoznanie názvoslovnej entity. Má to však aj svoje úskalia, nakoľko najviac jazykov sa úplne odlišuje tak v gramatike ako aj v slovnej zásobe, tým sťažujúc túto úlohu. Táto bakalárska práca skúma v troch jazykoch – angličtine, nemčine a maďarčine – ako informácie získané z písmen, slovných koreňov a slov ovplyvňujú vyzdvihovanie názvoslovných entitít.

Vďaka medzinárodnosti anglického jazyka vzniklo nespočetné množstvo sústav špecializujúcich sa na vyzdvihovanie názvoslovných entitít, ktoré pomocou kombinácie rozdielnych neurónových sietí dosahovali stále precíznejšie výsledky. Na vyhodnotenie takejto sústavy považujeme do dnešného dňa za smerodajnú F1-mieru, dosiahnutú na anglickej dátovej sade CoNNL 2003. Táto bakalárska práca berie za základ dva existujúce modely. Prvý skombinoval dlhodobo krátkodobú pamäť (LSTM) s konvolučnou neurónovou sieťou (CNN). Tento model ukázal, že vedľa vnorení slov sa dajú vytvoriť aj vnorenia písmen vďaka čomu nielen jednotlivé slová ale aj v nich obsiahnuté písmena môžu byť reprezentované číslami. Druhý model je kombináciou podmieneného náhodného pola (CRF) a dlhodobo krátkodobej pamäte (LSTM) skúša dosiahnuť, aby slová tvoriace jednotlivé vety neboli skúmané jednotlivo ale v závislosti od seba, tým berúc na ohľad kontext slov.

Model nachádzajúci sa v téze používa všetky tri hore spomínané moduly a vedľa dvoch hustých (slovné, charakterové) vnorení používa aj zriedkavé vnorenie. Cieľom je, aby vo forme rozdielnych testov vyšlo najavo nakoľko ovplyvňuje celkový výsledok množstvo a kvalita rôznych vnorení. Pritom cez mnou trénované slovné vnorenia skúšame vyriešiť takzvaný problém slov mimo slovnej zásoby, ktorý je zlovestnou výzvou rozpoznávania názvoslovných entít.

Pri budovaní modelu sme pomocou anglickej dátovej sady skúmali efektivitu rozdielnych neurónových sietí, hľadajúc ktorá sieť má najväčší vplyv na výkon modelu. Potom nasledovala jednotlivá optimalizácia hyperparametrov neurónových sietí, kde vyšlo najavo ktorými parametrami sa oplatí zaoberať a urobiť viac pokusov. V ďalšom kroku sa testovali vnorenia a analyzovali výsledky medzi mnou trénovanými a vopred trénovanými slovnými vnoreniami stiahnutými z internetu. Nakoniec sme sa pozreli či sa môže vylepšiť kvalita samotrénovaných slovných vnorení doplnením slov mimo slovnej zásoby rôznym spôsobom. Posledné dva kroky boli zopakované aj v prípade ďalších dvoch jazykov, kým pri hyperparametroch boli zmenené iba najvplyvnejšie.

Najlepšie dosiahnuté výsledky sme nakoniec porovnali s výsledkami iných moderných modelov ktoré pracovali s rovnakou dátovou sadou. Pri anglickom jazyku bolo našim cieľom priblížiť sa k výsledkom inšpiračného modelu. V tejto bakalárskej práci je najvyššia F1-miera pri anglických validačných dátach 94.82% pri použití predtrénovaného slovného vnorenia, kým pri inšpiračnom modeli je toto číslo 94.03%. Žiaľ pri anglickej testovacej dátovej sade dosiahol náš model 90.98% čím zaostal oproti pôvodnému modelu s 91.26%. Ostatné dva jazyky sa nachádzajú v dátovej sade WikiAnn, ktorá vyšla len nedávno, a preto ju málo modelov používa ako základ pri porovnávaní výsledkov. Pri nemeckom jazyku na testovacej sade pri použití samotrénovaného slovného vnorenia dosiahol systém 88.82%-né F1-mieru, po doplnení vnorení pre slová mimo slovnej zásoby sa výsledok zlepšil na 89.34%. V prípade maďarského jazyka bola situácia veľmi podobná, pred doplnením vnorení dosiahol systém na testovacej sade 92.44%-né F1-mieru, po vnorení týchto slov vyskočila hodnota na 93.04%.

Named Entity Recognition Exploiting Subword Information

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of P.h.D. Santosh Kesiraju. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

....
Patrik Dobrovodský
May 9, 2022

Acknowledgements

I am especially grateful for the assistance and guidance of my supervisor Ph.D. Santosh Kesiraju, without whom this thesis could not have been completed.

Contents

Chapter 1

Introduction

Despite rapid technological advances, most of the information is still in textual form. Today, even spoken information can be automatically transcribed into text. Yet, since no text follows the exact same structure, it is nearly impossible to create a system that can extract information – images, legal information, named entities – in an always consistent and reliable way. One of the main problems is the diversity of existing languages, all with their own distinct grammatical rules, making processing the data and finding the required information a difficult task.

Natural Language Processing (NLP) aims to solve certain problems in information retrieval and extraction. Its interest includes, but is not limited to: speech recognition, lemmatisation, and a subtask of information retrieval, Named Entity Recognition (NER). As the name suggests, the main purpose of NER is to find proper nouns in a given text and categorise them. For the English language, one might think of it as a trivial problem since every proper noun starts with a capital letter, but it is too insufficient for the categorisation itself, as there are many exceptions, such as the first letter of a sentence or entities composed of several words, only some of which are capitalised. In addition to this, there are languages that actually do not have a concept of capital letters, like most languages of East Asia.

In this work, our aim is to study and understand NER's statistical methods while harnessing the power of neural networks and putting an emphasis on the usage of subword units. Three different types of neural networks are combined for achieving positive results, namely a bidirectional Long Short-Term Memory network, a linear-chain Conditional Random Field, and finally a character-based Convolutional Neural Network. When it comes to subword units, the main focus is on word and character embeddings, sparse features, and the challenges they are able to solve. With all these tools, the goal is to create a system based on a once state-of-the-art system capable of recognising named entities supporting English next to two other Indo-European languages, specifically German and Hungarian, where one has to rely on additional subword information.

Chapter 2 introduces all the necessities, including the different approaches and challenges associated with NER. It also describes how different neural networks function and their role within NER. The chapter ends with an introduction of the state-of-the-art systems that were used as inspiration for the model proposed in this thesis. Chapter 3 describes the datasets and pretrained embeddings used for each language, and briefly explains how the custom-trained embeddings were created. Chapter 4 describes the proposed model and the tools and repositories used for its implementation. Chapter 5 summarises the experiments conducted with each language while evaluating the accuracy, and finally compares the results with various state-of-the-art systems.

Chapter 2

Formulation, approaches and challenges of Named Entity Recognition

Thanks to human and machine interaction and communication being a daily occurrence and a tremendous amount of textual data generated and shared on the Internet, Natural Language Processing (NLP) has become widely used in the world of data sciences. From automatic speech recognition to machine translation, this field contains numerous challenges dedicated to understanding the intricacies of interactions between humans and computers. One of these tasks, namely Named Entity Recognition (NER), is gaining popularity, as it is widely applicable for various information extraction problems. This chapter will briefly explain what NLP is and then dive deeper into how NER works and quickly explains the challenges that are present during the task and the approaches are used to solve them plus the metrics used for evaluation. Then this is followed by the exact formulation of the problem, followed by what neural networks are and how they are used inside the model. And finally, a paragraph about already existing state-of-art systems, that this work is based upon.

2.1 Natural Language Processing in a nutshell

There are two types of languages: formal (computer) languages and natural languages. Any language that has evolved naturally and is used for communication by humans like English, German, or Czech is considered to be a natural language. NLP studies said languages from both computational and linguistic standpoint and examines how computers can interact with them [13].

Elizabeth D. Liddy [14] defines NLP the following way: **“Natural Language Processing is a theoretically motivated *range of computational techniques* for analysing and representing *naturally occurring texts* at one or more *levels of linguistic analysis* for the purpose of achieving *human-like language processing* for a *range of tasks or applications*.”**
Parts of this definition can be elaborated on even more. *”Range of computational techniques”* means that there are various approaches existing when it comes to the type of language analysis. *“Naturally occurring texts”* means any language that humans use to communicate, both oral and written. The analysed text should not be created for analysis only,

but taken from actual use. The concept of *"levels of linguistic analysis"* indicates that we humans use multiple types of language processing to comprehend or produce languages. It is speculated that humans utilise every level, since they all convey a different type of meaning. The problem is that a given NLP system might use only one level, while others may apply multiple levels of linguistic analysis. This might cause confusion on what can we consider to be an NLP system, so specialist agreed that if a system uses any subset of the aforementioned levels of analysis, they are considered to be an NLP-based system. "Human-like language processing" tells us that it is a branch of Artificial Intelligence (AI). Even thought NLP depends on other disciples, it still aims to simulate human performance and can even outperform humans accuracy-wise in certain task, thanks to processing information in an entirely different way. "Range of tasks or applications" means that NLP itself is used to achieve a variety of tasks, be it Information Retrieval (IR), Machine Translation (MT) or Question-Answering [14].

2.2 Named Entity Recognition

Named Entity Recognition (NER) is one of the challenging subtasks of information extraction, where the goal is to filter both structured and unstructured documents and pinpoint phrases that refer to names of persons, locations or organisations. NER is an essential task and is considered to be a cornerstone of a NLP system. NER can be split into two tasks: the first is identifying proper names in text. The second part includes the classification into predefined categories such as names of persons, locations, organisations and other miscellaneous names. These proper names have been called Named Entities since the sixth Message Understanding Conference (MUC-6), which contributed significantly to the research of this area[17]. A typical tagged text made by a NER system looks like the following:

> In [$_{\text{LOC}}$ **Kansas City**], [$_{\text{PER}}$ **Juan Guzman**] tossed a complete-game six-hitter to win for the first time in over a month and lower his league-best [$_{\text{MISC}}$ **ERA**] as the [$_{\text{ORG}}$ **Toronto Blue Jays**] won their fourth straight, 6-2 over the [$_{\text{ORG}}$ **Royals**].

Contrary to part-of-speech tagging, where every word gets its own tag, NER finds and labels spans of text where the difficulty stems from the ambiguity of segmentation, meaning we have to set the boundary of what is considered a named entity and what is not. A high percent of the words will not be named entities, that is to be expected. The type ambiguity can cause problems as well. For example, the word "Chelsea" can refer to an individual, an area in West London, or even a football club [12].

The standard labelling format for NER is BIO tagging [28]. This helps us label the words with tags that contain both boundaries and the type of the named entity. The BIO tagging uses 3 characters: B for the beginning of a span of interest, I for words still inside the span, and O for words that are outside any span. There are also variations like the IO tagging, which neglects the B tag, thus losing some accuracy and the BIOES tagging introducing the E tag for ending a given span and the S tag for a single word span [12].

Word	IO Tag	BIO Tag	BIOES Tag
Martinez	I-PER	B-PER	S-PER
begins	O	O	O
play	O	O	O
against	O	O	O
Ruxandra	I-PER	B-PER	B-PER
Dragomir	I-PER	I-PER	E-PER
of	O	O	O
Romania	I-LOC	B-LOC	S-LOC
.	O	O	O

Table 2.1: Differences between IO, BIO, BIOES tagging illustrated on a sentence taken from the training set of CoNLL 2003, adapted from [12]

The first approaches to NER were mainly rule-based. They consist of a set of rules for named entity extraction plus a lexicon containing domain specific words (gazetteers) which are applied to the text with the help of an extraction engine. The gazetteers and rule sets were either handmade by humans or extracted from custom-made examples. Rule-based systems precision is mostly reliant on how comprehensive the gazetteers and rules are, which explains why they are effective only on narrow domains. Manually incorporating more words into the lexicons, makes it a very expensive effort for a slight improvement [19].

With more data resources becoming available, the popularity of statistical NLP methods has risen, incentivising NER research to focus on a more data-driven approach, which has greatly reduced the cumbersome human work needed to create rule sets and lexicons. Statistical NER requires two components:

1. A text corpus with annotated words also called training data, as seen in Table 2.1.
2. A statistical model trained to fit this training data.

A statistical model contains parameters that map events of a language to the probability of a predefined label. If a model would be trained on the sentence "Chelsea bought Lukaku." the model would try to predict how likely each word can be marked with one of the predefined labels. There are multiple choices for classification, one of them is to classify every word on its own, which is usually done by supervised learning models but this would mean that the dependency between words would fail to be taken into account. Instead, NER is considered to be a sequence labelling task in which the system tries to predict the labels of an entire sequence of tokens, thus modelling the dependency existing between tokens. This way, in the previous example, the word "Chelsea" is easier to label correctly when the entire sequence is known, as it reduces the problem of disambiguation [19].

2.2.1 Challenges in Named Entity Recognition

There are various challenges one has to solve when dealing with NER, like the previously mentioned word ambiguity. In addition to that, there are abbreviations such as writing

WHO instead of Word Health Organisation or just simple grammatical mistakes found in the text corpora. One of the major challenges when it comes to datasets or even real data is the so-called out-of-vocabulary (OOV) problem. To put it simply, let us suppose that a training set has a thousand-word vocabulary, while the test set contains another thousand. Each word that is found in the training set, but not present in the test set is considered to be an OOV word.

There are multiple methods to solve the OOV problem. One of the easiest techniques is simply assigning random embeddings to each word that does not have one. This solution is mediocre at best because assigning random numbers and hoping that the model can optimise them during training is not reliable. A different approach would be to use the neighbouring words as a reference to create a word embedding for the OOV word. This is done by averaging two or three neighbouring embeddings from both sides to acquire a better representation of the unknown word. This works better most of the time, as words usually depend on each other to form a coherent sentence, but it still heavily relies on the optimisation of the training process. One of the best methods is to use external tools like fastText[1] to train word vectors on a large amount of data, which results in a model capable of creating word embedding for words it has never before seen. The accuracy of the embeddings is still dependent on words seen during the training of the external model, but with a large enough training data finding similar expressions to deduce embeddings for OOV words. All of these techniques are going to be tested to see which method performs the best on real data.

The second big challenge is tied more so to the language itself, as different languages have distinct grammatical rules, syntaxes, and morphologies. This thesis deals with two Indo-European languages, namely English and German, where the former is classified as an analytic language and the latter is an inflected language. Besides these two, there is also Hungarian, which is an agglutinative language. The similarities and differences that are considered important for NER are demonstrated with short sentences in Table 2.2.

English	German	Hungarian
I love Czechia.	Ich liebe Tschechien.	Szeretem Chehországot.
I live in Czechia.	Ich lebe in Tschechien.	Chehországban élek.
I travel to Czechia.	Ich fahre nach Tschechien.	Chehországba utazom.
I come from Czechia.	Ich komme aus Tschechien.	Chehországból jövök.
I write about Czechia.	Ich schreibe über Tschechien.	Chehországról írok.

Table 2.2: Illustrating the similarities and differences between English, German and Hungarian languages.

It is immediately obvious that the biggest difference is in the usage of prepositions. Both English and German use prepositions, which ensures that the named entity itself does not change its form, as seen in the examples “Czechia” and ”Tsechien” always stay the same. For Hungarian, it is the exact opposite, as agglutinative languages keep adding affixes to the stem of the word to change its meaning. So, while the stem “Csehország” is consistent, adding different affixes like “-ban” or “-ból” change both meaning and form of the word. This causes a substantial problem when it comes to word embeddings, because each affix connected to the word, means another entry added into the word embeddings. This results in having the same named entity unnecessarily repeated multiple times without a

[1] https://fasttext.cc/

meaningful change in its numerical representation. The solution to this problem is to try to detect the stem of the word instead of trying to create word embeddings for each variation. This is why character-based Convolutional Neural Networks are commonly used to handle this problem, as they look at the word character by character. When it comes to words like “Csehország” and “Csehországot”, which differ only in two characters, the Convolutional Neural Network detects this and creates two very similar character embeddings for them. Therefore, the ideal way to solve this problem is combining both word and character embeddings to get the best possible numerical representation for each word.

2.3 Evaluation metrics

During training it is important to measure how well the system performs, so at regular intervals the model evaluates how accurate it is in predicting the correct tags for all sentences in the development set and when the model performs the best a model checkpoint is set up to be later used on the test set. However, simply calculating the amount of predictions guesses and dividing it by all of the predictions made would give a poor representation of how accurate the model really is, since correctly predicting the O tag is also counted, which is the most frequent tag. A better method is to evaluate only the named entities, which is why the F-score is used instead for measuring how well the system does. The rest of this section is based on [33]. To be able to measure the classification’s quality, both correct and incorrect examples have to be taken into account, meaning there are four possible outcomes of the recognition, which is easily demonstrated on the following confusion matrix:

	Recognized as Positive	Recognized as Negative
Positive	True Positive	False Negative
Negative	False Positive	True Negative

Table 2.3: A confusion matrix for classification. Taken from [33].

When it comes to NLP, the interest lies in finding the positive cases; thus two metrics named precision and recall have been defined with the following formulas:

$$\text{precision} = \frac{\text{number of true positives}}{\text{number of true positives} + \text{number of false positives}} \tag{2.1}$$

$$\text{recall} = \frac{\text{number of true positives}}{\text{number of true positives} + \text{number of false negatives}} \tag{2.2}$$

The F-score is the harmonic mean of precision and recall, the only thing left is adding the variable β which is used to choose which of these two measurements is preferred. In case of β being 1 it is balanced, if $\beta < 1$ recall is considered to be the more important metric in the other case the precision is favoured.

$$F\text{-}score = \frac{(\beta^2 + 1) \cdot precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{2.3}$$

2.4 Formulation of the problem

This section is adapted from [34]. The goal of NER is to predict the output variables $\mathbf{y}$ given a set of input variables $\mathbf{x}$. To be more precise, $\mathbf{x}$ is a vector representing each word in a sentence, while $\mathbf{y}$ is a vector containing the corresponding tag for every word. For example, there are N input and output pairs $\mathbf{x}_i, \mathbf{y}_i \, \forall i = 1 \dots N$ within the input sequence $\mathbf{x}$. To make it simpler, let us work only with one sentence, which has T number of tokens. This way, the two vectors can be written as:

1. Input sequence tokens :
$$\mathbf{x_i} = [x_1, x_2, \dots, x_T]$$
2. Output labels :
$$\mathbf{y_i} = [y_1, y_2, \dots, y_T]$$

There are various approaches and models that exist when it comes to NER. The easiest way to look at the problem would be to simply consider it as a classification problem and try to classify each word independently. This technique is non-traditional *naive Bayes classification*, and as the name suggests, it is based on the Bayes theorem, but the emphasis is on the *naive* assumption that all the input tokens are independent. It is also non-traditional, as in the case of traditional naive Bayes y is not a sequence. A probabilistic formulation would be the following:

$$\begin{aligned} p_\theta(\mathbf{y} \mid \mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})} &= \frac{p(\mathbf{x} \mid \mathbf{y}) p(\mathbf{y})}{\sum_{\bar{\mathbf{y}} \in \mathcal{Y}} p(\mathbf{x} \mid \mathbf{y} = \bar{\mathbf{y}}) p(\bar{\mathbf{y}})} \\ &= \frac{\prod_{t=1}^{T} p(x_t \mid y_t) p(y_t)}{\sum_{\mathbf{y} \in \mathcal{Y}} \left[\prod_t p(x_t \mid y_t = \bar{y}_t) p(y_t = \bar{y}_t) \right]} \end{aligned} \tag{2.4}$$

Here, $\mathcal{Y}$ marks all the possible tags. The equation above can be illustrated graphically, which makes understanding the essence of the non-traditional naive Bayes easier. Figure 2.1 illustrates how it would look as a simple directed graph.

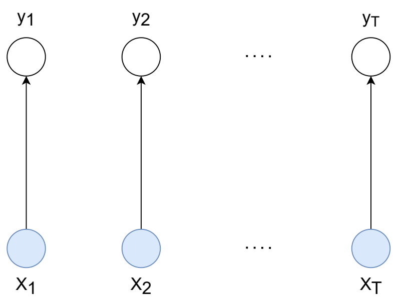

Figure 2.1: Directed graph of the naive Bayes classifier. Neither the input tokens, nor the output labels are dependent on each other, each and every one of them is classified on its own.

As mentioned in Section 2.2 it is important to model the dependency between words. This is why a better approach for this task is the Maximum Entropy Markov Model (MEMM), also known as Logistic Regression. Unlike the naive Bayes classifier, it assumes that the input sequences are conditionally dependent on each other. The formulation of the logistic regression is the following:

$$p_\theta(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{T} p(y_t|\mathbf{x}) = \prod_{t=1}^{T} \frac{\exp\left\{\phi(\mathbf{x}, y_t)\right\}}{\sum_{\bar{y}_t \in \mathcal{Y}} \prod_t \exp\left\{\phi(x, y_t = \bar{y}_t)\right\}} \tag{2.5}$$

Here in the case that ϕ would be a linear regression then this would be considered a MEMM, but in the case that it being, for example, a BiLSTM then it cannot be considered one. The MEMM uses the whole input sequence $\mathbf{x}$ as a monolithic variable. This means that the output is dependent on the whole sequence instead of only the current word. Figure 2.2 shows this dependency:

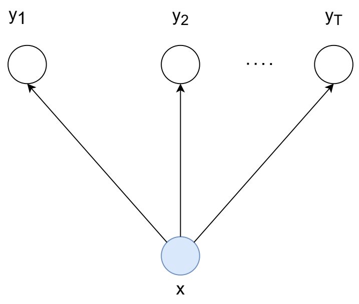

Figure 2.2: Directed graph illustrating linear regression.

The final improvement to this model would be creating a dependency between the output tags as well. An obvious upgrade would be to make each output label dependent on the previous label. This would allow for better recognition, especially when it comes to named entities consisting of multiple words. This is precisely what the linear chain Conditional Random Field is capable of doing, which is formulated in the following way:

$$\begin{aligned} p_\theta(\mathbf{y} \mid \mathbf{x}) &= p(y_1 \mid \mathbf{x}) \prod_{t=2}^{T} p(y_t \mid x, y_{t-1}) \\ &= \frac{\exp\left\{\phi(x, y_t) + T(y_t, y_{t-1})\right\}}{Z(x)} \end{aligned} \tag{2.6}$$

The training of a linear-chain CRF is usually done with maximum likelihood seeking θ^*, which can be imagined as the combination of the model's parameters meaning the weights, biases, embeddings and transitional probabilities. In the above equation, T stands for the transition scores, and Z(x) is called the partition function. All of these parameters will be explained later in detail in Section 2.5.5. So, the main objective is to maximise this likelihood:

$$\theta^* = \text{argmax}_\theta \prod_i p_\theta(\mathbf{y_i}|\mathbf{x_i}) \tag{2.7}$$

Technically, to get the exact same results, one can also just minimise the loss as follows:

$$\text{argmin}_\theta - \prod_i p_\theta(y_i|x_i) \tag{2.8}$$

Figure 2.3 demonstrates how the linear-chain Conditional Random Field would look like:

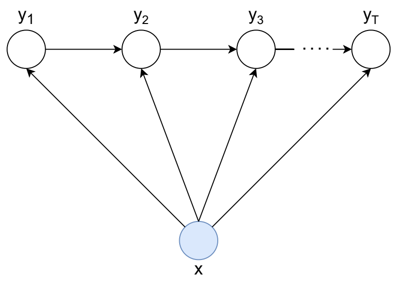

Figure 2.3: Directed graph illustrating the linear-chain Conditional Random Fields.

2.5 Approaches to Named Entity Recognition

2.5.1 Word embeddings

This subsection is based on [4].One of the significant breakthroughs for NLP problems was the usage of word embeddings, allowing words that are similar in meaning to have resembling representations. Word embeddings are real vectors in a predefined vector space where every individual word gets its own vector, thus creating a densely distributed representation. These vectors can have tens, or even up to hundreds of dimensions, which is still significantly less than the thousand of dimensions used for methods like one-hot encoding. The distributed representation tries to capture the meaning of words, so words that are used in a similar fashion are closer to each other. This strongly differs from the bag-of-words model where the usage of the words is not taken into account, so different words all get their own representation.

There are three main ways that word embeddings can be learned from text. The first one is called an embedding layer, which needs the document to be prepared, so then every word can get one-hot encoded. Then the vectors are randomly initialized, while the size of the vector space varies based on the model. This so-called embedding layer then gets fed into the neural network, aiding the supervised learning process. This method requires a big amount of training data and is quite slow, and can be used only with the specific data it was trained on.

The second is a statistical method called Word2Vec [18] specialised for developing pre-trained word embeddings from a text corpus. It introduced a deeper analysis of word vectors and created stronger ties between the representations of words. For example, if we take the word “King” and subtract “manliness” from it, and then we add “womanliness” the result would be the word “Queen” because these embeddings learned to capture the relationship between man and woman. Word2Vec introduced two learning models as well: one being the Continuous Bag-of-Words (CBOW) model, which learns by using the context to predict the current word. The other is the Continuous Skip-Gram Model doing the exact opposite, predicting the context based on the current word. These approaches have a huge advantage, namely their low time and space complexity allows learning embeddings from bigger corpora, often with billions of words.

The final algorithm is the Global Vectors for Word Representation (GloVe) [25] refining the Word2Vec method, applying matrix factorisation techniques that excel at using global text statistics with the aforementioned context-based learning. With the construction of a word co-occurrence matrix and the help of statistics extracted from the corpus, it manages to create a learning model resulting in a higher quality word embeddings.

2.5.2 Neural network

This section has been adopted from [40]. Artificial neural networks (in simple terms, neural networks) take the human brain as an inspiration, where billions of neurons work tirelessly to process information in parallel. Neural networks are made up of layers: an input layer of nodes, one or multiple hidden layers of nodes, and an output layer of nodes. Every node in one layer is connected to the adjacent one, and every connection is associated with a numerical value called weight. If we have a neuron i in the hidden layer, we can write its output as:

$$h_i = \sigma(\sum_{j=1}^{N} V_{ij} x_j + T_i^{hid}) \tag{2.9}$$

where:

$$
\begin{aligned}
h_i &= \text{output of neuron i} \\
\sigma &= \text{activation function} \\
N &= \text{number of input neurons} \\
V_{ij} &= \text{weights} \\
x_j &= \text{inputs fed to the input layer} \\
T_i^{hid} &= \text{threshold of the hidden neurons}
\end{aligned}
$$

The intention behind the activation function is to confine the value of a neuron, so that the neural network is not impaired by divergent neurons. One common activation function is the sigmoid function, but there are also others, such as the arc tangent or the hyperbolic tangent. With this architecture, any computable function can be approximated to an arbitrary precision.

The other crucial component of a neural network application is training, which is performed by feeding the neural network data from the *training set*. The purpose of training is to adjust the weights of the connected neurons, which is done by minimising the *loss function*. One of the common functions is *Mean Squared Error*, which takes the sums of the squared differences between the expected outputs and the actual outputs of the neural network. In the case of this work, if the CRF is used, then the *Negative Log Likelihood Loss* is used to minimise the loss, in other case the *Cross-Entropy* loss is used instead, which aims to maximise the posterior probability of the correct label, given the input data and the model parameters. To help experiment with different architectures, another data set can be applied to the already trained network, simply named as *validation set.* The one producing the best results is then picked up and used later on. Finally, the last independent data set named *test set* is utilised to determine the level of performance, showing how well the neural network performs on a previously unseen set. On the one hand, if the performance is good on the test set, it means that the neural network is able to generalise well. On the other hand, if the performance is poor on the test set but at the same time good on the training set, that could boil down to the following:

1. The neural network did not generalise well enough.

2. The neural network has overfit on the training data.

3. The test data can be from an entirely different domain.

To avoid either of these possibilities, the training set has to be big enough, so the neural network could memorise characteristics embedded in it, but at the same time not too big as to avoid making the network waste precious resources on fitting the noise. Therefore, a thoughtful representation of the data is vital to creating a successful implementation of neural networks.

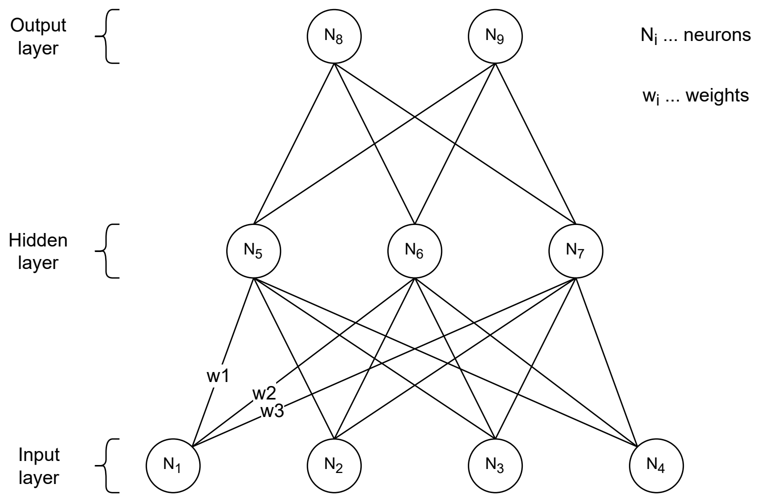

Figure 2.4: Possible architecture of a feed forward neural network

2.5.3 Long Short-Term Memory

Human thoughts have persistence, if we are reading some text we do not start thinking from scratch every time but try understanding the next word based on the previous one read. The same cannot be said for the aforementioned neural networks. Recurrent neural networks (RNN) solve this issue by having loops in them, thus allowing information to persist. It can be imagined as multiple copies of the same network, and every one of them passes a message to their neighbour. In theory, carefully choosing the right parameters means that RNNs are capable of handling long-term dependencies. In practice, this is not the case, this is why a special kind of RNN has been designed explicitly to solve this dependency problem. The Long Short-Term Memory Network (LSTM) was created by Hochreiter and Schmidhuber [11] to remember information for a long period of time[6].

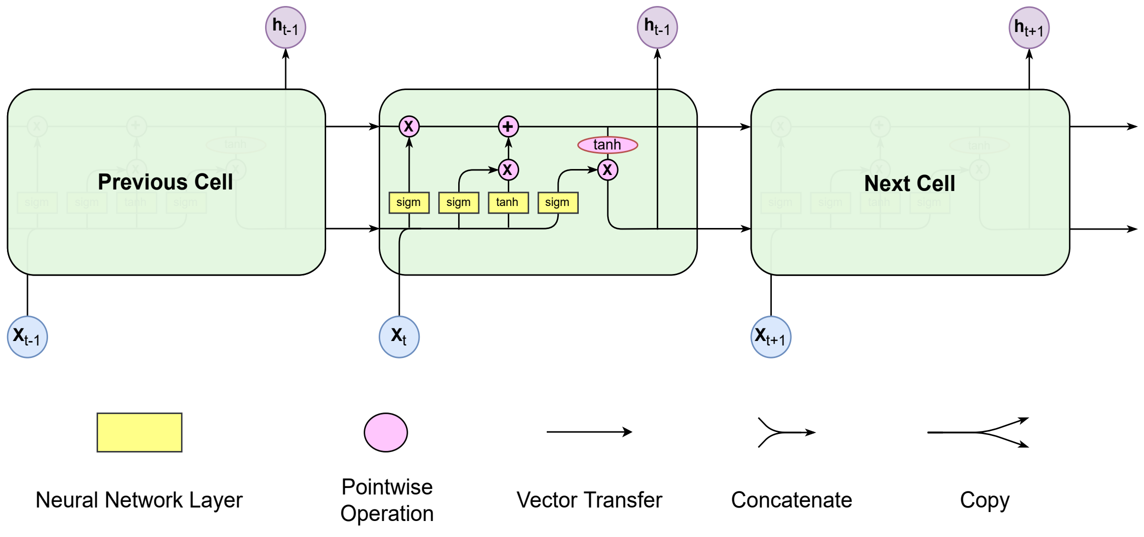

Figure 2.5: Repeating modules in an LSTM network, inspired by [6].

LSTM layer is described by [10] in the following way: “An LSTM layer consists of a set of recurrently connected blocks, known as memory blocks. These blocks can be thought of as a differentiable version of the memory chips in a digital computer. Each one contains one or more recurrently connected memory cells and three multiplicative units – the input, output and forget gates – that provide continuous analogues of write, read and reset operations for the cells. ... The net can only interact with the cells via the gates.”

One of the most significant improvements of the LSTM is the bidirectional LSTM. Conventional RNN has another weakness, it only makes use of previous context and does not take future context into account. Bidirectional RNN solves this issue by having two separate hidden layers processing the input data from both directions: one layer in the forward direction, the other in reverse. For every time step, the output of the BiLSTM layers is generated by combining the cell memory vectors of the two LSTM from both sides. This means that the contextual information of the whole sequence is taken into consideration [39].

2.5.4 Convolutional Neural Network

Convolutional Neural Networks (CNNs) were inspired by mammal’s cortical region, to be more exact small areas of cells in the cortex that are sensitive to specific areas in the field of vision. CNN is made up of different collection of layers, each different in their functionality, so they are divided into three categories: convolution layer, pooling layer and fully connected layers [31].The rest of the subsection is based on [42].

The convolution layer is a crucial part of CNN that is responsible for feature extraction, which combines convolution operation with an activation function. The most frequently used activation function is the rectified linear unit (ReLU), but the sigmoid or the hyperbolic tangent functions are viable as well. The convolution operation uses a kernel, which is a small array of numbers that is applied across the input. This input is also an array of numbers called tensor. Both kernel and input can be either one-dimensional (audio, text), two-dimensional (image) or even three-dimensional (video, 3D image). A Hadamard product between every element of the kernel and the tensor is computed and summed at every location of the tensor. The output values make up the output tensor also called feature map. This method is repeated with multiple kernels to create arbitrary amounts of feature maps, each representing a different characteristic of the input tensor.

One problem with this operation is that it does not allow the centre of any kernel to overlap with the outermost element of the input tensor, thus reducing both the height and the width of the feature map. This problem is counteracted by using zero padding, simply adding rows and columns of zeros to each side of the input tensor. Without it, each consecutive feature map would shrink in size after a convolution. The final parameter of convolution is the stride, describing the distance between two successive kernel positions. A stride greater than 1 is used in case feature maps need to be downsampled. During training, the main goal of the convolution layer is the identification of the best kernels based on the training data. Only the values of the kernel are learned automatically during training; the aforementioned hyperparameters, like kernel size, number of kernels, padding, and stride are set before the training starts.

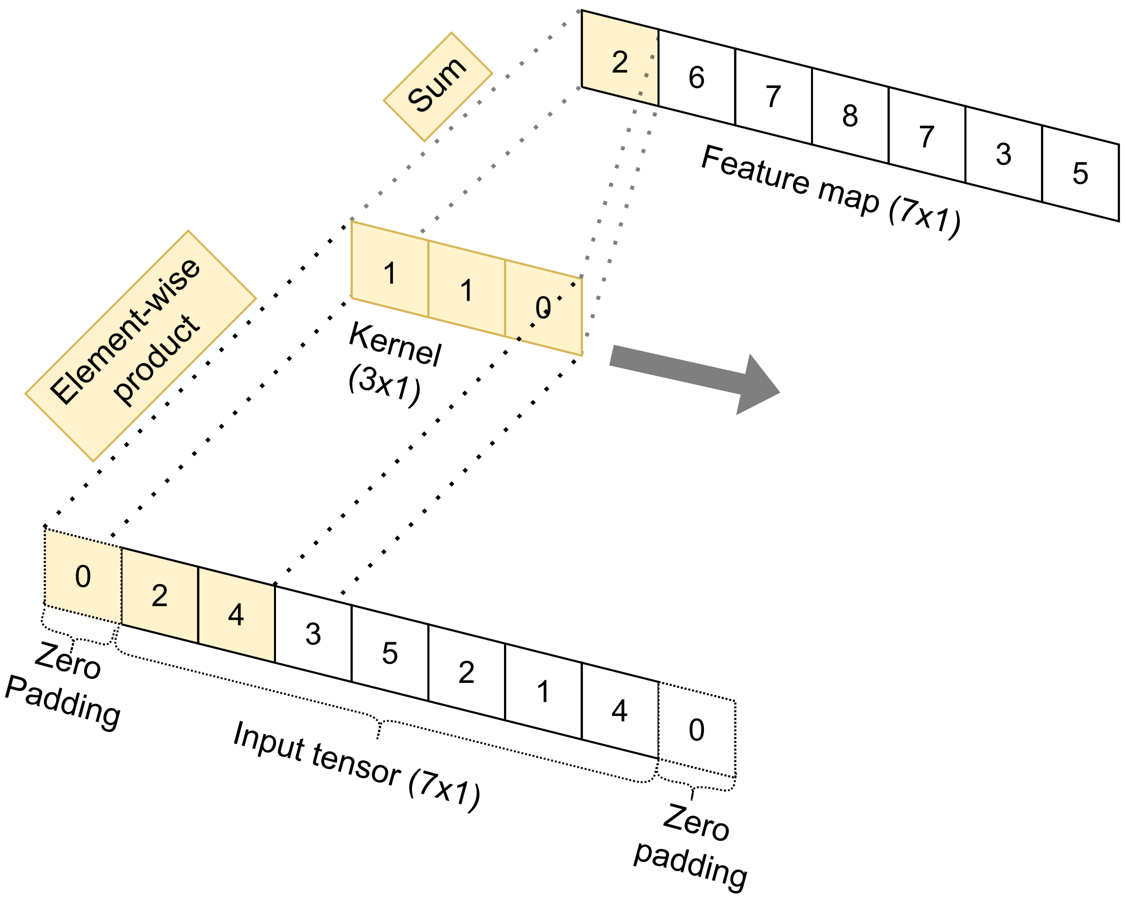

Figure 2.6: Example of a convolution operation on a one-dimensional tensor with zero padding and kernel size 3×1 with a stride of 1

The pooling layer reduces the dimension of feature maps (downsampling), thus decreasing the amount of subsequent learnable parameters. During training, the pooling layer is unable to learn anything, but has almost identical hyperparameters as the previous layer, namely: filter size, stride, and padding. One established operation is max pooling, which extracts patches from the input feature maps and returns the maximum value from each patch. For image classification a commonly used filter size is 2 with a stride of 2, which achieves downsampling the feature map's in-plane dimension by a factor of two, while keeping the depth dimension unchanged. For the creation of character embeddings as there is no downsampling needed, a stride of 1 is preferred instead.

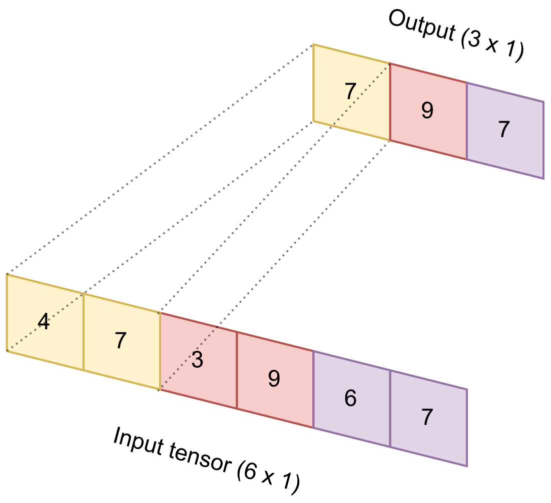

Figure 2.7: Example of max pooling with filter size of 2 and with a stride of 2.

The output of the final convolution or pooling layer is transformed into a one-dimensional array, which then serves as the input for one or multiple fully connected layers, which is a simple feed forward neural network. After each fully connected layer, there is an activation function like the previously used ReLU. The final fully connected layer most of the time has the same amount of output nodes as the number of classes in the given classification task. Following the final fully connected layer, an appropriate activation function is needed based on the task at hand. For a classification problem with multiple classes, the softmax function is applied, which normalises the output real numbers to the target class probabilities, so the values are between 0 and 1 and their sum equals to 1.

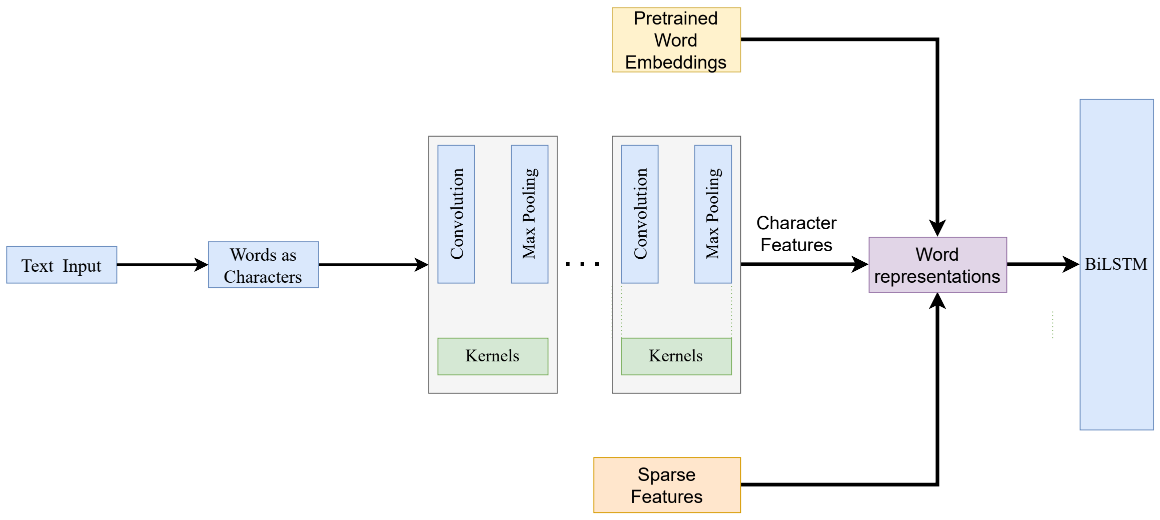

Figure 2.8: Example of how the thesis model will use the Convolutional Neural Network to extract character features.

2.5.5 Conditional Random Field

Conditional Random Field (CRF) combines discriminative classification with graphical modelling, which allows the creation of models having multivariate outputs $\mathbf{y}$ while using a considerable amount of input features $\mathbf{x}$ for prediction [34]. They are all undirected graphical models, but the most commonly used graphical structure is the linear chain, which resembles a finite-state machine, which is therefore qualified for sequence labelling [24]. The rest of this section is adapted from [38]. For a sequence labelling problem, we want to solve the conditional probability $p(\mathbf{y} \mid \mathbf{x})$ where $\mathbf{y}$ is a sequence of labels and $\mathbf{x}$ is a sequence of input vectors. Considering this as a regular classification problem, we can multiply the probability of each item at k-th position, which results in:

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) &= \prod_{k=1}^{\ell} P(y_k|x_k) \\ &= \prod_{k=1}^{\ell} \frac{\exp\left(U(x_k, y_k)\right)}{Z(x_k)} \\ &= \frac{\exp\left(\sum_{k=1}^{\ell} U(x_k, y_k)\right)}{\prod_{k=1}^{\ell} Z(x_k)} \end{aligned} \tag{2.10}$$

Here, ℓ means the length of the given sequence. There are multiple reasons for using the exponential, as it helps avoiding problems like underflow and having to work with negative numbers, and most importantly it allows to create a log-linear formulation. $U(x, y)$ is called unary scores because it gives a score to a label y given the x vector at time step k. The unary function can be based on simple rules or statistics that can provide a score for the probability that the input x_k belongs to label y_k. In the case of the thesis model, the easiest way is to think of them as the outputs of the BiLSTM. $Z(x)$ is named partition function, which is used for normalisation since the results will be probabilities.

After this, new learnable weights are added to represent the probability of label y_k being followed by y_{k+1}. This way, there is a dependency between each successive label, which justifies the name linear chain CRF. The previous probability is multiplied by $P(y_{k+1}|y_k)$ and by using the exponential properties the result is:

$$P(y|X) = \frac{exp\left(\sum_{k=1}^{\ell} U(x_k, y_k) + \sum_{k=1}^{\ell-1} T(y_k, y_{k+1})\right)}{Z(X)} \tag{2.11}$$

where:

$U =$ Unary scores meaning how likely is the label y_k given x_k as the input

$T =$ Transition score meaning the probability of the label y_k being followed by y_{k+1}

$Z =$ Partition function used for normalization

The problem lies within the partition function, which would need to cycle through and sum over every possible combination the label set can have at each time step. This would be computationally difficult, so it is solved by using dynamic programming with an algorithm called the Viterbi algorithm, but its specific explanation is beyond the scope of this work.

2.6 State-of-art Name Entity Recognition system

This work is based on two different NER systems. The work of Chiu and Nichols [5] combines both word and character embeddings next to lexicons to achieve a high accuracy on the CoNLL 2003 English dataset. Their model extracts features from every word, which then gets fed into a stacked bidirectional LSTM network and gets turned into probabilities with the help of different layers such as linear and log-softmax. With the help of a convolution and max pooling layer, they create feature vectors for each character using the character embedding. To make this easier, zero padding is used to make every word have the same length. For word embeddings, they experimented with both GloVe and Word2Vec embeddings, but their best performing model uses the 50 dimensional embeddings created by [8]. Additionally, a lexicon was compiled from named entities from DBpedia, to help encode words found both in the corpus and the lexicon into their BIOES annotated form. With these features, their model has managed to achieve a 94.03 F1-score on the CoNLL 2003 development set.

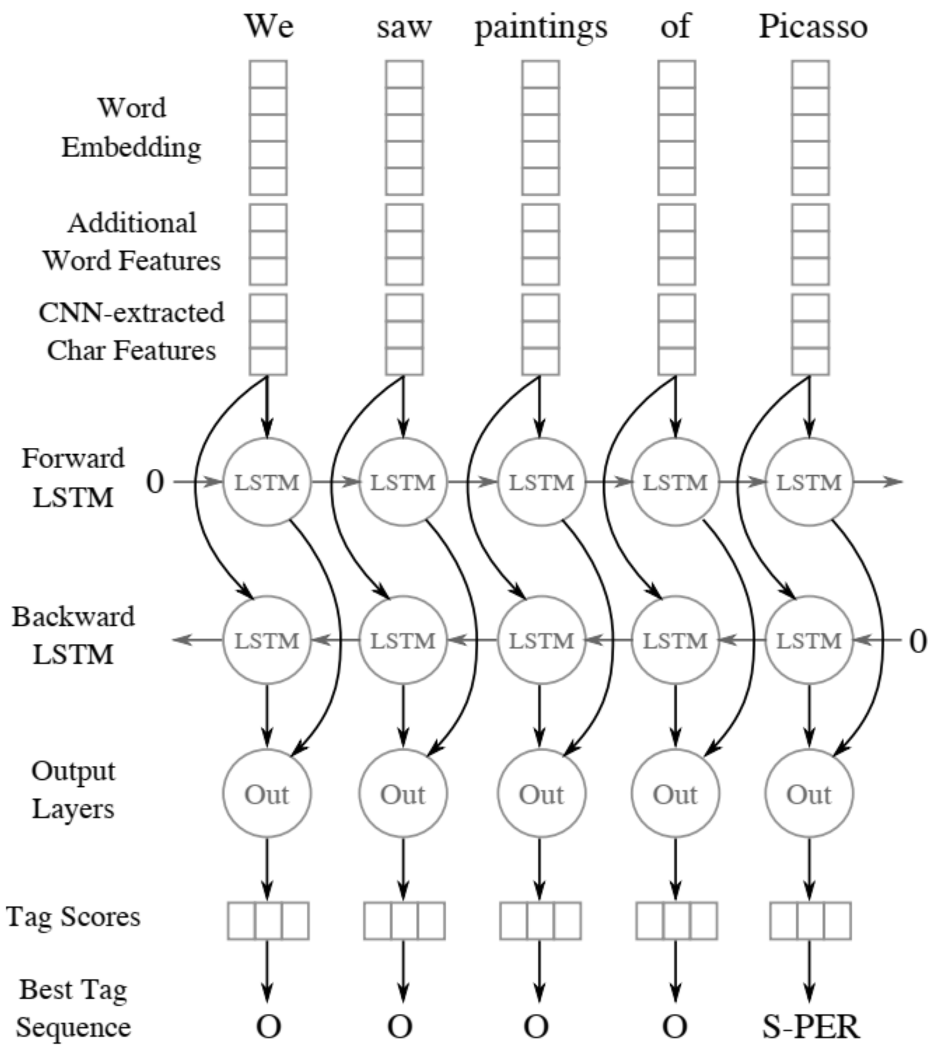

Figure 2.9: The model of Chiu & Nichols. Picture from [5].

The other work from Abujabal and Gaspers [1] concentrates on subword units and uses a bidirectional LSTM-CRF network. The CRF part replaces the normally used softmax layer, which turns the LSTM output into probabilities. For every subword unit, there is a bidirectional LSTM creating a word embedding based on the characters, bytes, and phonemes that are making the current word up. This allows to mitigate the problem with out-of-vocabulary words not being present in the pretrained word embeddings. Their datasets contain requests made to voice-controlled gadgets in four languages, including English, French, German, and finally Spanish. Their best performing model combined the character and phoneme units to achieve a 94.02 F1-score on a real-world English dataset.

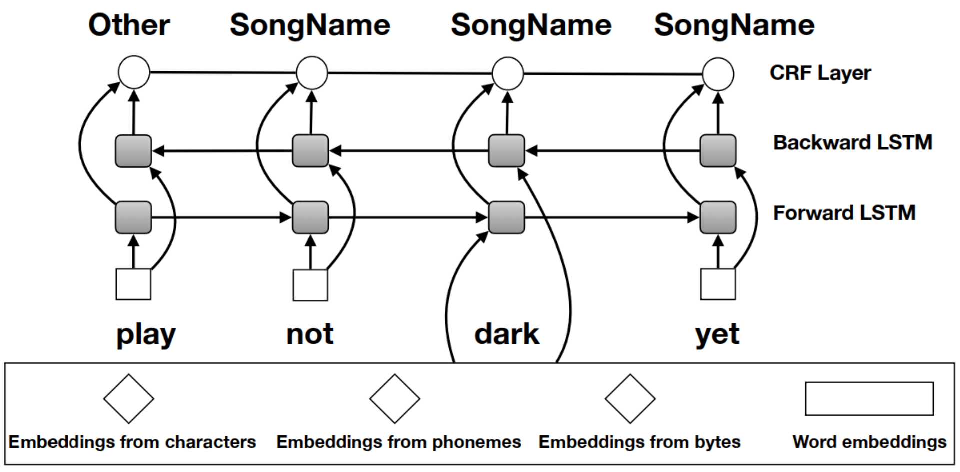

Figure 2.10: The model of Abujabal & Gaspers uses a bidirectional LSTM layer with the CRF layer used for decoding. For every word, their model learns the embeddings from characters, phonemes and bytes, meaning the usage of dedicated embeddings is not required. Source: [1].

Chapter 3

Used datasets and pretrained word embeddings

When training a neural network, a large amount of data is needed as it attempts to learn semantic representations of words in context. There is a large amount of textual data available for use, and finding correctly annotated data for Named Entity Recognition(NER) in many languages has become easier than ever. The same applies to word embeddings, as there are many tools that allow their creation given enough text resources. For NER in English, one of the regularly used datasets for performance benchmark is the CoNLL-2003 Shared Task [37] dataset. When it comes to other languages, the more recent WikiAnn dataset [22] contains annotated sentences for around 200 languages from which Hungarian and German will be used. Word embeddings are just as important for the success of NER, so two variants will be mentioned in this chapter. Word embeddings that were already trained on some data and available for download, called pretrained embeddings, and custom-trained embeddings, which were created by downloading different corpora and using tools to create embeddings for each language.

3.1 CoNLL-2003 dataset

The CoNLL-2003 dataset covers two languages: German and English. The English data is created from 1 year's worth of stories from the Reuters Corpus, which contains news articles. The data contain 4 types of named entities: persons, organisations, locations and miscellaneous marked as PER, ORG, LOC and MICS, respectively. The dataset uses the BIO tagging to distinguish multi-word named entities.

	Training	Development	Test
Sentences	14 987	3 465	3 683
Tokens	204 567	51 578	46 666
Named entities	23 499	5 942	5 648
LOC	7 140	1 837	1 668
MISC	3 438	922	702
ORG	6 321	1 341	1 661
PER	6 600	1 842	1 617

Table 3.1: Statistics of the CoNLL-2003 English data.

3.2 WikiAnn dataset

Wikipedia is currently one of the largest multilingual datasets that contains labels specifically for NER and supports a myriad of languages. The original WikiAnn dataset [22] uses Wikipedia articles to create annotated data for 282 languages, using external knowledge bases and neural networks. Their work was refined by [27], thus producing balanced splits for 176 languages, with the BIO tagging and supporting 3 types of named entities: locations, persons and organisations. Out of these 176 languages, this work uses the German and Hungarian datasets.

WikiAnn - Hungarian	Training	Development	Test
Sentences	20 000	10 000	10 000
Tokens	180 653	88 792	90 302
Named entity tokens	28 507	14 086	14 163
LOC	11 566	5 611	5 671
ORG	8 070	4 025	3 982
PER	8 871	4 450	4 510

Table 3.2: Statistics of the Wikiann - Hungarian dataset

WikiAnn - German	Training	Development	Test
Sentences	20 000	10 000	10 000
Tokens	195 387	97 805	97 646
Named entity tokens	27 643	13 818	13 868
LOC	9 778	4 968	4 961
ORG	8 575	4 281	4 157
PER	9 290	4 569	4 750

Table 3.3: Statistics of the WikiAnn - German dataset

3.3 Pretrained word embeddings

When it comes to pretrained word embeddings, the two most popular choices are created through Word2Vec or GloVe, as already discussed in 2.5.1. For the English dataset, two pretrained embeddings made by GloVe[1] were experimented with. One of them has been made out of Wikipedia dumps and Gigaword5, and contains 6 billion tokens with a 400 thousand word vocabulary, while the other one from 2 billion tweets, which adds up to 27 billion tokens with a 1.2 million word vocabulary. For the German language, GloVe embeddings created from Wikipedia articles were acquired from Deepset[2]. For all three languages, word embeddings from fastText[3] were used, which were created based on the work of Bojanowski et al.[3]. They were trained on Wikipedia data, using a skip-gram model.

[1]https://nlp.stanford.edu/projects/glove/
[2]https://www.deepset.ai/german-word-embeddings
[3]https://fasttext.cc/docs/en/pretrained-vectors.html

3.4 Data for training word embeddings

Thanks to the large amount of text resources available and tools such as fastText[4] and GloVe[5] existing, creating word embeddings is now an easily achievable task. The process starts with the acquisition of a large amount of textual data for the three languages, from the website Opus[6]. The tables below give information about the corpora used for each language. All text data are concatenated, then sentences are tokenised followed by word tokenization with the help of the NLTK library[7]. These tokenised data are then used to train a skip-gram with subword information. The first output is the word embeddings themselves, the other is the model itself, that can be used to infer embeddings for out-of-vocabulary(OOV) words via the nearest neighbour query. As mentioned in Section 2.2.1 there are multiple strategies for obtaining embeddings for OOV words, and the nearest neighbour was one of them. The experiments carried out using different methods are presented later in Chapter 5.

English	
Corpus	Tokens
WikiMatrix [32]	1 000 000 000
EU bookshop [36]	380 200 000
Wikimedia [36]	349 200 000
TildeMODEL [30]	131 400 000
Europarl [36]	33 000 000
TED2020 [29]	5 900 000
News Commentary [36]	4 900 000
Multi UN [36]	4 700 000
Global Voices [36]	1 300 000
Sum	1 910 600 000
Unique Words	5 062 525

German	
Corpus	Tokens
WikiMatrix	443 100 000
Wikimedia	11 000 000
TildeMODEL	108 800 000
EU bookshop	337 400 000
Europarl	30 500 000
TED2020	5 400 000
News Commentary	5 000 000
MultiUN [36]	4 300 000
Global Voices	1 300 000
ParaCrawl[8]	450 700 000
Open Subtitles[9] [15]	4 000 000
Sum	1 397 900 000
Unique Words	8 477 877

Table 3.4: Statistics of the English and German corpora used for training of embeddings.

[4] https://fasttext.cc/docs/en/unsupervised-tutorial.html
[5] https://github.com/stanfordnlp/GloVe
[6] https://opus.nlpl.eu/
[7] https://www.nltk.org/
[8] https://paracrawl.eu/
[9] https://opus.nlpl.eu/OpenSubtitles-v2018.php

Hungarian	
Corpus	Tokens
WikiMatrix	99 700 000
Wikimedia	2 300 000
TildeMODEL	43 000 000
EU bookshop	14 800 000
Europarl	12 300 000
TED2020	4 900 000
ParaCrawl	3 800 000
Open Subtitles	500 000
Sum	181 300 000
Unique Words	3 538 450

Table 3.5: Statistics of the Hungarian corpora used for training of embeddings

Chapter 4

Design and implementation of the proposed system

With the theory discussed above, the next step is to show how the proposed Named Entity Recognition(NER) model is designed. This chapter gives a brief overview of how the proposed model block schema looks like and what important parameters does it use, and going over which technologies were used to create the system.

4.1 The proposed Named Entity Recognition system

As this thesis is inspired by the two previously mentioned systems in Chapter X the first obvious choice is to include a bidirectional Long Short-Term Memory (BiLSTM) network as it serves as the basis for both of these models. As the goal is to utilize subword units, the addition of the character-based Convolutional Neural Network's(CNN) improves performance especially when used together with word embeddings as it reduces problems when it comes to out-of-vocabulary words. And finally a Conditional Random Field layer which makes the predictions for a word dependent on every immediate neighbour, thus incorporating context for every prediction. With all this in mind, I decided to create the system shown in Figure 4.1.

This system exploits subword information with the combination of three different vectors, starting with loading in the pretrained word embeddings (dense features) giving a multidimensional numerical representation for each word. This is followed by the creation of the sparse features, which take into account only three properties per word, namely: if it starts with a capital letter, if it is a digit, or if it contains a digit. For every matching property, it gets a 1 otherwise a 0, creating a vector mostly containing zeros, hence the name sparse representation. And finally, the features extracted from the characters via the character-based CNN. The pretrained word embeddings, sparse features, and representations from the character-based CNN are concatenated and fed into the BiLSTM memory network with the goal of acquiring tag scores. The Hidden to Tag layer is there in respect to the implementation as it helps transform the output of the BiLSTM into the correct shape, meaning it can transition forward into the layer with the Conditional Random Field (CRF). During training the CRF layer will be used to calculate the negative log-likelihood loss, which will be used for the backpropagation process, which simply means updating the parameters of the model which include the weights of the neural network and the transition probabilities of the CRF. When it comes to the evaluation part, the Viterbi algorithm is used, to get the most likely sequence of tag scores.

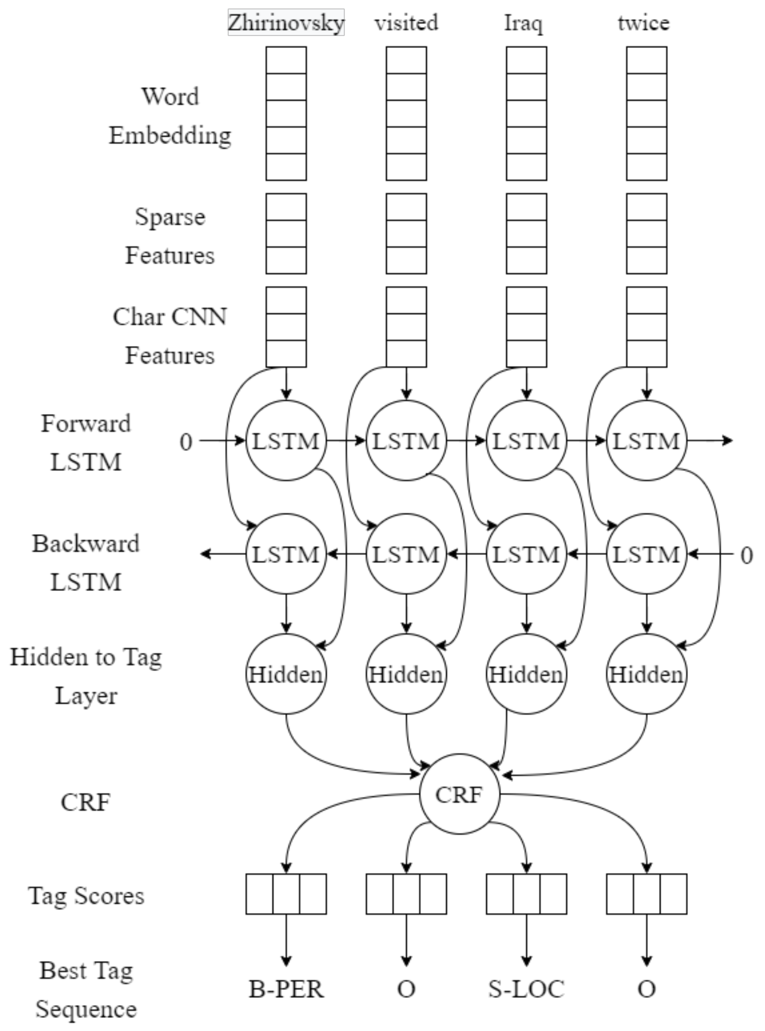

Figure 4.1: The proposed NER model, containing three different modules : Convolutional Neural Network (Char CNN), a Long Short-Term Memory (LSTM) network and finally the Conditional Random Field (CRF).

Naturally, as the system is made up of smaller functioning units, it is unavoidable to have multiple important parameters where each of them has a significant say in what the final results will be. Some of the important hyperparameters and the results achieved with them are presented later in Chapter 5. There are various hyperparameters just for the character-based Convolutional Neural Network alone, for example:

- Character embedding dimension – deciding the size of the vector representing each character in a word.

- Number of character filters (kernels) and the convolution width – indicating how many times the convolution and max pool layers are repeated in the network, and how many filters are applied at each convolution operation.
- The other hyperparameters are related to the convolution operation, assigning the kernel's size, the stride used and if padding is necessary as seen already in 2.6

The following important hyperparameters are connected to the training itself:

- Dropout – zeroes some of the elements in a tensor, to avoid overfitting.
- Batch size – meaning how many sentences does the system go through before updating the model parameters. For example, having 1000 sentences with a batch size of 50 means that there will be 20 batches with 50 sentences each.
- Epochs – indicate how many times does the program go through the whole training set, simply said going through every batch.
- Optimizer – for obtaining the gradient and updating the model parameters.
- Learning rate – connected to the optimizer, decides how quickly does the model adapt to the task, and how big of a change is allowed for the weights during backpropagation.

4.2 Implementation tools

I chose Python[1] as the implementation language as it is very versatile and universally used to tackle different machine learning tasks. It also supports object-oriented programming, making it easier to build the application as individual blocks working together.

With that in mind, the next step is choosing a machine learning library to fit the needs of the system. One of the best open source libraries, PyTorch [23], created by Facebook's Artificial Intelligence Research lab, has become really popular thanks to its intuitive nature and well-documented functionality. The library was written mostly in C++, thus ensuring high performance. It provides not only computation with tensors, which can be accelerated by the graphics card for faster and more efficient calculations, but also contains frameworks for different deep neural networks. It also provides the ability to convert NumPy arrays into PyTorch tensor, and support creating custom datasets by subclassing the library's dataset class and implementing methods for indexing and returning the length.

There are countless implementations of NER systems online. As such, I have taken inspiration from cswangjiawei's public GitHub repository[2] for the creation of the model itself and how the character-based Convolutional network is built up. For the Conditional random field the code has been taken from NCRF++ repository[3] which is an open source sequence labelling toolkit. When it comes to the evaluation the CoNLL 2000 Shared Task included an evaluation script in Perl language, and the same script[4] has since been remade in Python language supporting every kind of dataset, provided that it is made with IOB2 or IOBES tagging.

[1]https://www.python.org/
[2]https://github.com/cswangjiawei/pytorch-NER
[3]https://github.com/jiesutd/NCRFpp/tree/master/model
[4]https://github.com/sighsmile/conlleval

Chapter 5

Experimentation and comparison with state-of-the-art systems

Experimentation is the most important part of this thesis, as the goal is to achieve better results by combining two existing Named Entity Recognition (NER) models and, more importantly, to study and analyse the NER approaches for OOV words. Detailed analysis is performed to see how the addition of neural networks and Conditional Random Field (CRF) affects the precision of the system. Testing will take place on different datasets for English, German, and Hungarian languages, but to avoid repetition, hyperparameter tuning and detailed analysis are only included once on the English dataset for the sake of comparison. This chapter contains experiments showing how influential adding different modules is to the system when working with the English dataset, then how the best hyperparameters were found during the continuous testing, how exactly did the system fare with the other two languages, and finally a comparison with other state-of-the-art systems.

5.1 Experimentation with neural networks

To start with the experimentation, the bidirectional Long Short-Term Memory (LSTM) network has been implemented, followed by the character-based Convolutional Neural Network (CNN) and the Conditional Random Field(CRF). As mentioned, each of the experiments uses the CoNLL 2003 English dataset and one of the available pretrained word embeddings. I decided to use the 100 dimension GloVe embeddings downloaded from the Stanford[1] website, as it has been trained on both news and Wikipedia articles. When it comes to the out-of-vocabulary words, the model initialises them to zeros and relies on the training to correct their value. As the hyperparameters of the model have not yet been optimised, all experiments are going to use the same parameters for easier comparison. Table 5.1 contains all hyperparameters used to train the model, while Table 5.2 displays the hyperparameters used by the CNN only.

[1]https://nlp.stanford.edu/projects/glove/

Word embedding dimension	LSTM hidden dimension	Batch size	Learning rate	Dropout	Epochs	Optimizer	Random number seed
100	200	10	0.01	0.1	100	Stochastic Gradient Descent	32

Table 5.1: Hyperparameters of the model used throughout the early testing. The goal is to add each neural network and different features to create the proposed model and then fine-tune all of its hyperparameters.

Character embedding dimension	Number of kernels	Kernel size	Padding	Stride
50	1	2	1	1

Table 5.2: The initial hyperparameters of the character-based CNN.

As the goal is to assess how precise the system is, after every fifth epoch, the model runs the evaluation process on both the development and the test set. The result of each evaluation is saved and later used to create a graph to compare how well the model performs on these sets. The highest F1-score achieved in the development set is taken as the basis for the final comparison. So, even though the model could theoretically achieve a higher score for the test set, the one shown in the results is based on the highest score of the development set. With the parameters mentioned above and the evaluation process, experiments were conducted with the goal of finding out how the combinations of different neural networks perform. Table 5.3 contains each combination and the performance achieved on the development set and the test set.

NER model	Development set			Test set		
	Precision	Recall	F1-score	Precision	Recall	F1-score
BiLSTM	92.53	92.19	92.36	87.69	87.50	87.59
BiLSTM + CNN	93.21	94.19	93.7	88.62	90.01	89.31
BiLSTM + CRF	94.24	92.73	93.48	89.73	88.05	88.88
BiLSTM + CNN + CRF	94.22	94.45	94.34	90.30	90.86	90.58
BiLSTM + CNN + CRF + Sparse	94.30	94.35	94.32	90.64	90.88	90.76

Table 5.3: Comparison between the performance of different neural networks using the same parameters. The final entry uses sparse features to enrich the embeddings used with word-specific information.

From the results, it is easy to assess that the character-based CNN greatly improves the performance when it comes to both the development and the test sets. It is not a surprise because when it comes to word embeddings, 2235 words are OOV ones, which represents exactly 8.87 % of the overall vocabulary that the dataset has. This means that, for these OOV words, the addition of character embeddings enhances the model's ability to predict these words correctly even when there is literally no numerical representation at the beginning. The high recall also supports this claim, as it indicates the model's ability to distinguish positive samples.

When it comes to the CRF, the recall is almost as low as it was during the BiLSTM run. This is understandable, as the problem with the OOV words is not solved with the addition of this module. However, the precision of this model is relatively much higher. This can be explained by the CRF making the labels depend on each other, making the identification of named entities consisting of multiple words more accurate, thus increasing positive classifications.

The combination of these two modules achieves a better balance between these two metrics. The addition of sparse features gives minor gains, as it only helps to distinguish words starting with a capital letter and to decide whether a word is a digit or just contains a digit. Figure 5.1 illustrates the evaluation metrics in the development and test sets as the training progresses with the model using all the modules found in the last row in Table 5.3.

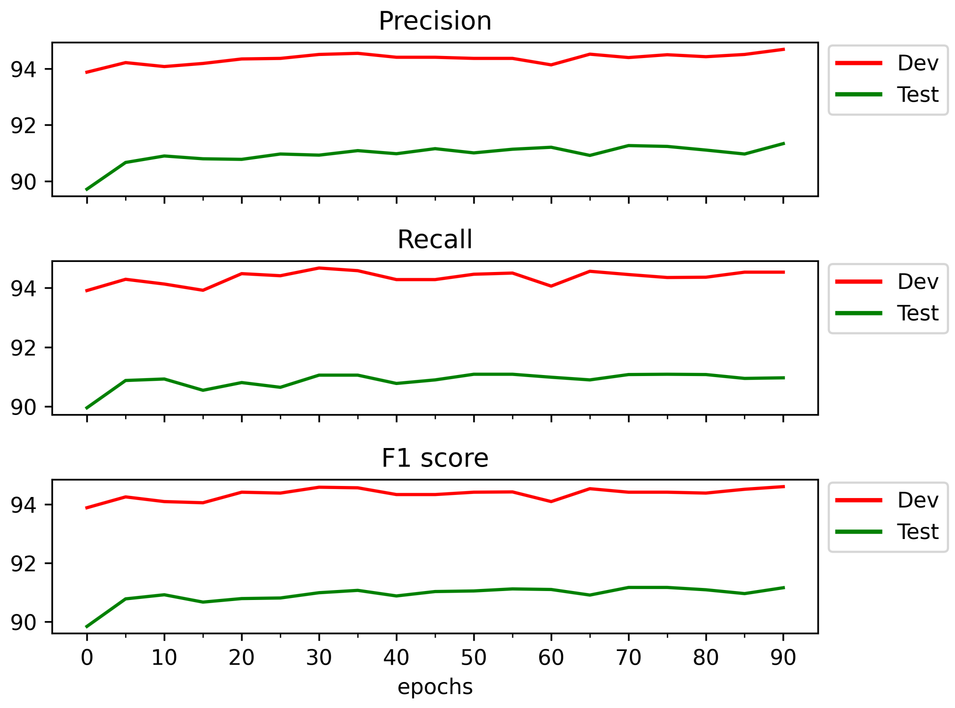

Figure 5.1: The evaluation process of the model using BiLSTM + CNN + CRF + Sparse during the training. The highest values are achieved between the 30th and 40th epoch after which the model starts to stagnate. The highest F1-score achieved on the development set was 94.32%, while on the test set it was 90.76%.

5.2 Optimization of hyperparameters

Having a functional model ready, the next step is all about trying to find the most optimal settings that produce the best results. As the model has a large number of different hyperparameters, it is crucial to find the ones that are the most influential when it comes to achieving the highest possible F1-score. The best possible way to find out, is to conduct many experiments while only changing one aspect of the model to see how beneficial the change actually is. I decided not to experiment with every possible hyperparameter, so the following values also found in Table 5.1 and Table 5.2 will remain consistent throughout the testing phase:

- Epochs – the length of the training process should always remain consistent, to make comparison easier, and to give each model the same chance to achieve the highest accuracy.
- Optimizer - Pytorch offers a large number of optimisation algorithms[2], testing them all would take a long time while offering diminishing returns.
- Kernel size – while the kernel size could be constant for each convolution, in the final version this hyperparameter linearly increases with the number of kernels in the model. So, if the number of kernels is five, the first configuration will have a kernel size of one, while the last one will have a kernel size of five.
- Padding and stride – these two hyperparameters are mostly used for downscaling, which is not necessarily used when working with text, and thus there is no need to change them.

With this in mind, the experimentation started with the remaining hyperparameters. As the LSTM plays an important role in the model, I decided to conduct the first experiments with its hidden dimension. Figure 5.2 shows the highest F1-score achieved with different values on the test and development sets. From the results, it is clear that when it comes to the development set, the 150 hidden dimension does slightly better than the others, but at the same time it does a bit worse than the 100 dimensional when it comes to the test set. As the goal is to achieve the best score on the development set, a decision has been made to rather use the 150 dimension variant.

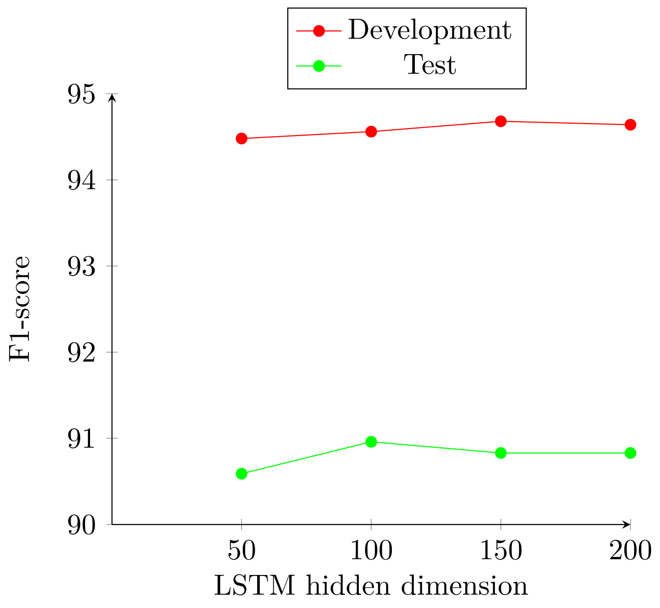

Figure 5.2: The comparison between the highest F1-score while using 4 distinct hyperparameters for the LSTM hidden dimension. The difference between the development set score is almost negligible, but the F1-score for the test set can be almost half a percent.

[2]https://pytorch.org/docs/stable/optim.html#algorithms

The next hyperparameter on the list that might have a greater impact on the F1-score is the batch size. Most commonly, powers of two are used as values for batch size. From the results seen in Figure 5.3, it is clear that a smaller batch size is preferable, as 16 and 32 did the best considering both sets. Although 16 performs better on the test set, 32 still performs slightly better when it comes to the development set.

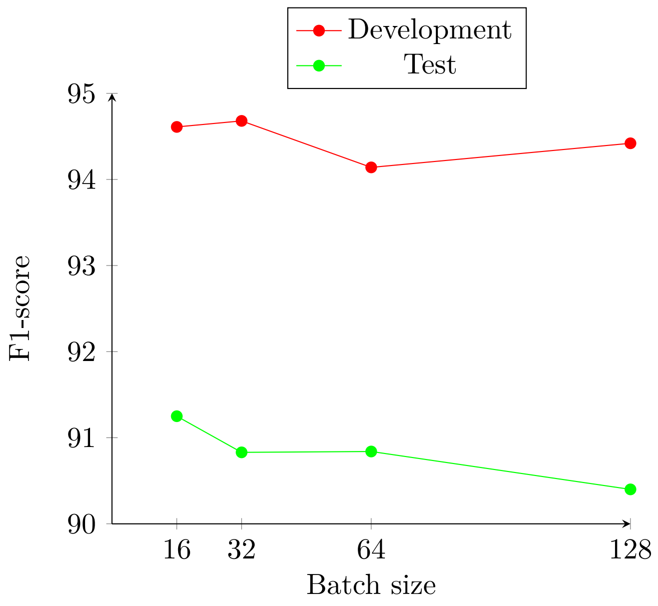

Figure 5.3: Comparison of model's highest F4 score with batch sizes being the powers of two. Lower numbers produce better results especially on the development set.

Since the scores of 16 and 32 are not far from each other, I decided to take into account another aspect of the training, time. To be more precise, the question is how long one epoch of training takes with a given batch size. Smaller batch sizes obviously mean longer training epochs, as going through all sentences takes a longer time. Table 5.4 also confirms this and shows a substantial difference in time between the highest and lowest batch sizes. Even between the values 16 and 32, there is exactly half a minute of difference during training and even more when it comes to evaluation. This is why in the long run it is beneficial to choose 32 as the final batch size as it performs better on the development set and is twice as fast during training.

Batch size	Average Training Epoch Length	Average Evaluation Epoch Length
16	$64.12 \approx 64$ sec	$92.10 \approx 92$ sec
32	$33.64 \approx 34$ sec	$59.26 \approx 59$ sec
64	$27.39 \approx 27$ sec	$53.21 \approx 53$ sec
128	$24.81 \approx 25$ sec	$52.05 \approx 52$ sec

Table 5.4: The average time of one epoch during training and evaluation ni seconds. Higher batch size means faster training as more sentences are being processed at the same time.

For the next hyperparameters, I considered having a closer look at how the word and character embedding size influences the model's output. The model is tested with the pre-trained GloVe embeddings, which have four different dimensional versions. From Figure 5.4

it is obvious that the 100-dimensional embedding performs exceptionally well in the development set compared to its counterparts and reasonably well on the test set to be chosen as the final hyperparameter.

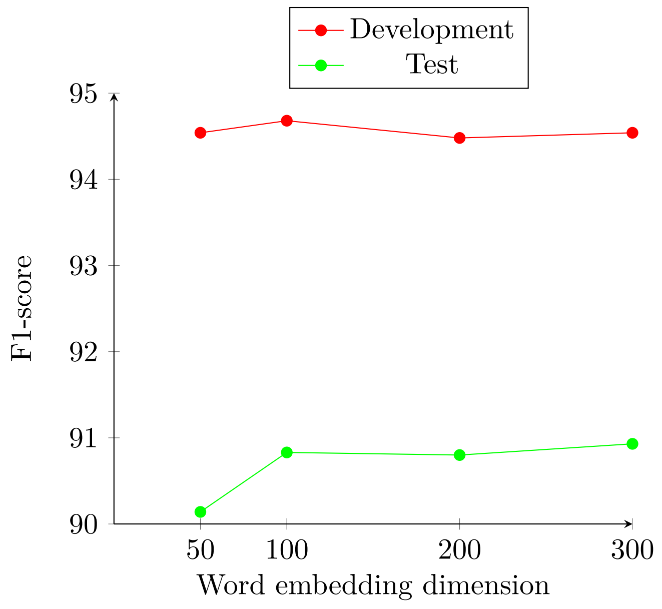

Figure 5.4: Comparison of model's highest F1-score while using English pretrained GloVe embeddings containing the same words with different embedding dimensions.

For character embeddings, I decided to hover around the value that Chiu and Nichols [5] used as their final hyperparameter, which is 53. But the results seen in Figure 5.5 actually prove that in this case values smaller than 50 perform better on the development set, while being quite close to or even better than the high values for the test set. The differences here are not substantial. between the worst and the best performer on the test set is a mere 0.06 difference. So, even though the value 30 is the worst performer on the test set, it is still chosen as the final hyperparameter due to its performance on the development set.

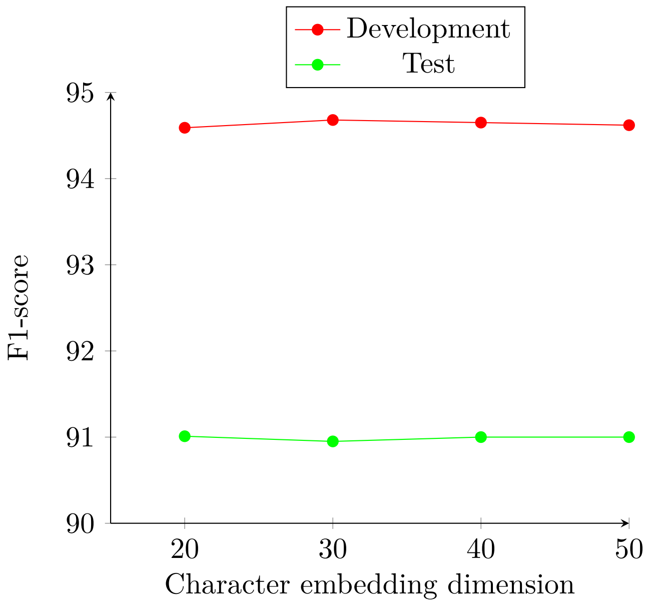

Figure 5.5: The difference of the highest F1-score with using different output sizes for the Convolutional Neural Network.

The same experiments were conducted for each hyperparameter, which was not mentioned at the beginning of Section 5.2. Table 5.5 contains the range and results of hyperparameter optimisation. Using optimised parameters means that the system should achieve the highest F1-score on the development set. Figure 5.6 shows how the evaluation process looks when using the highest performing hyperparameters.

Hyperparameter	Range	Final
Convolution width	[1, 3]	**3**
Character embedding dim	[20, 50]	**50**
Word embedding dim	[50, 300]	**100**
LSTM hidden dimension	[50, 200]	**150**
Learning rate	$[10^{-2}, 10^{-1.7}]$	**0.015**
Batch size	[16, 64]	**32**
Dropout	[0.2, 0.5]	**0.3**

Table 5.5: The range of hyperparameter values on which the experiments were carried out. The final column contains the values achieving the best results on the English dataset, and were used with the other two languages as well.

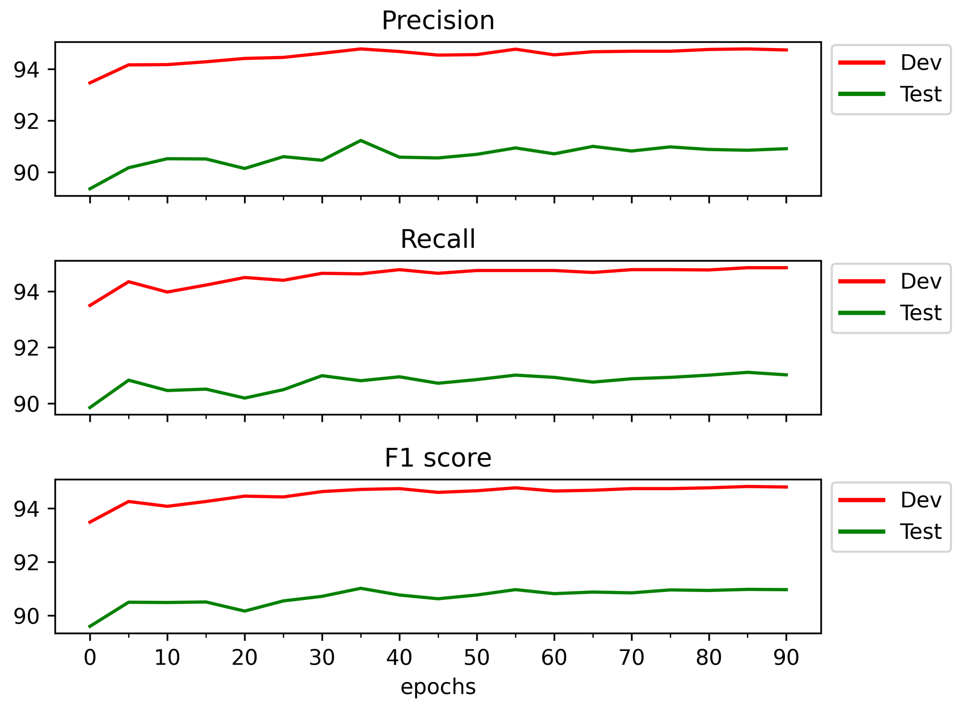

Figure 5.6: The evaluation of the model with the final hyperparameters. It is clear that there is less fluctuation compared to the results of the initial model seen in Figure 5.1. The highest F1-score for the development set is 94.82% compared to the initial 94.32%. For the test set, the best score is 90.98% next to the initial 90.76%.

5.3 Experimentation with word embeddings

As all the hyperparameters are optimised, the next objective is to try to determine how different word embeddings influence the model's performance. As mentioned in Chapter 3, there are pretrained and custom-trained word embeddings available for the English language. For easier comparison, every embedding is changed to be exactly 100 dimensional. There are three pretrained embeddings acquired from the Internet and one that was custom-trained with the use of fastText. The results of their performance can be found in Table 5.6.

English embedding	Trained on	Development set			Test set		
		Precision	Recall	F1-Score	Precision	Recall	F1-Score
Pretrained GloVe	Wikipedia	94.79	94.85	94.82	90.85	91.11	90.98
	Twitter	93.94	93.87	93.91	89.87	89.64	89.75
Pretrained fastText	Wikipedia	94.17	93.79	93.98	89.65	89.13	89.39
Custom-trained fastText	Custom	94.25	94.01	94.13	90.53	90.00	90.26

Table 5.6: The highest performance on the CoNLL 2003 English dataset with the optimised model, using four different word embeddings. The custom-trained embeddings have been trained on the corpora found in Table 3.4.

With these results, the next objective is to find out if the number of out-of-vocabulary (OOV) words actually affects how well the model performs. Table 5.7 shows the statistics for each embedding file and exactly how many OOV words they contain.

English embedding	Trained on	Unique words	CoNLL 2003 Dataset			
			Perfect match	Case match	OOV words	OOV %
Pretrained GloVe	Wikipedia	400 000	11 415	11 656	2235	8.83
	Twitter	1 193 514	10 872	9785	4649	18.37
Pretrained fastText	Wikipedia	2 518 768	10 848	11 770	2688	10.62
Custom-trained fastText	Custom	2 933 851	23 301	188	1817	7.18

Table 5.7: The number of out-of-vocabulary (OOV) words found in different English word embeddings. Even though the custom-trained word embeddings have the least amount of OOV words, they are still not performing as well as the pretrained GloVe embeddings trained on data from Wikipedia.

From the results, it is evident that the model works better when using the GloVe embeddings. Interestingly enough, even though the Twitter GloVe embedding has the highest OOV percentage, it still has a barely worse performance on the development set than the pretrained fastText data. So, at first glance, it might seem that having a lower OOV percentage means nothing when it comes to performance. Up to this point, OOV words were not taken into account in any way, instead they were represented with zero embeddings and left to the model to update them during the training process. However, as mentioned in Section 2.2.1 there are multiple ways of solving the OOV problem, which will be tested on the custom-trained dataset with the goal of seeing how these different approaches influence performance. The first approach, reffered to as "Zero" only fills up the OOV word embedding with zeros. This is followed by the "Random" method, whereas the name suggest the embeddings are uniformly distributed random numbers from the interval $[-1, 1)$. Next up, the "Average" approach which during the first epoch finds each OOV word in a sentence, takes the embeddings of the two neighbouring words from both sides (if they exist) and averages them. And finally, the "Nearest neighbour" technique, which uses the custom-trained fastText model to create embeddings for the OOV words that are actual named entities based on subword information encountered during its training. The results produced by each approach can be seen in Table 5.8.

English OOV approach	Development set			Test set		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Zero	94.25	94.01	94.13	90.53	90.00	90.26
Random	94.25	94.04	94.15	90.16	90.01	90.09
Average	94.36	94.06	94.21	89.74	89.24	89.49
Nearest neighbour	94.36	94.28	94.32	90.15	90.26	90.21

Table 5.8: Comparison of four different approaches to solving the out-of-vocabulary (OOV) problem with the English custom-trained word embeddings.

On the basis of their performance, the zero and random approaches do not differ much, but it is clear that using zero embeddings guarantees better performance on the test set. Unfortunately, even when using the nearest neighbour query and essentially having no named entities as OOV words, the performance is still not on par with the more sophisticated GloVe embeddings. At a first glance, it seems that word embeddings infered with the nearest neighbour approach are clearly superior to the other methods, as it uses subword information combining the embeddings of similar words to produce the best possible numerical representation. Another thing worth investigating is the amount of OOV named entities that were actually predicted correctly while using different methods. The statistics of correct predictions of OOV named entities can be seen in Table 5.9. The results show that even though all of the OOV named entities are filled in with the nearest neighbour approach, that does not mean that every word necessarily gets the correct representation, but it is still more reliable than either the random or the average approach.

English OOV approach	Development set			Test set		
	OOV named entities	Correct %	Incorrect %	OOV named entities	Correct %	Incorrect %
Zero	206	83.98	16.02	236	77.12	22.88
Random	206	83.01	16.99	236	74.15	25.85
Average	206	83.01	16.99	236	74.15	25.85
Nearest neighbour	0	88.35	11.65	0	79.24	20.76

Table 5.9: Statistics on the prediction of the OOV named entities in the Enlgish test and development sets. While the nearest neighbour approach essentially creates embeddings for all the OOV named entities, the same entities are checked to see if the dedicated embeddings helped to predict them correctly.

5.4 Experiments with the German language

With the experimentation on the English dataset completed, there are still two languages left to run the same tests on. Although the model was optimised, during previous experimentation, a decision was made to still run tests with some of the most influencing hyperparameters, such as the hidden dimension of the LSTM network and the batch size. In some cases, when the hidden dimension is changed, the model actually performs better, which is always indicated below the results. The best performance of the model with the use of three different word embeddings can be found in Table 5.10.

German embeddings	Trained on	Development set			Test set		
		Precision	Recall	F1-Score	Precision	Recall	F1-Score
Pretrained GloVe	Wikipedia	88.63	87.07	87.85	88.56	87.22	87.88
Pretrained fastText	Wikipedia	89.14	87.65	88.39	89.04	87.63	88.33
Custom-trained fastText	Custom	89.34	87.97	88.65	89.47	88.18	88.82

Table 5.10: Comparison of the best performance using the WikiAnn German dataset, with three different word embeddings. Both fastText embeddings performed better with the LSTM hidden dimension being 200 and the batch size being 32. The custom-trained embeddings were trained on the corpora found in Table 3.4.

The custom-trained word embeddings clearly outperform the other two pretrained embeddings. But even though it is the best performer, the results are very far from the performance seen in the English language. This is probably due to an unusual property of the German language, namely that every noun starts with a capital letter. This means that it is harder to differentiate named entities from normal nouns, and that the model does not gain any meaningful advantage by having sparse features available. Yet again, it is important to analyse whether the number of OOV words actually sway the performance in a significant way. Statistics related to OOV words can be seen in Table 5.11.

German embeddings	Trained on	Unique words	WikiAnn German dataset			
			Perfect match	Case match	OOV words	OOV %
Pretrained GloVe	Wikipedia	1 309 281	8015	41 251	11 437	18.84
Pretrained fastText	Wikipedia	2 274 727	8491	44 129	8083	13.32
Custom-trained fastText	Custom	5 771 728	53 953	50	6700	11.84

Table 5.11: The number of out-of-vocabulary (OOV) words found in different German word embeddings. The custom-trained embeddings have been trained on the biggest amount of data, that is why it has the least amount of OOV words.

For the German language, there seems to be a correlation between the percentage of OOV words and performance. The next step is to once again see if different OOV approaches on custom-trained word embeddings have any positive impact on overall performance. Table 5.12 shows how performance differs with the use of different approaches.

German OOV approach	Development set			Test set		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Zero	89.34	87.97	88.65	89.47	88.18	88.82
Random	89.04	88.00	88.52	89.38	88.43	88.90
Average	89.50	88.25	88.88	89.59	88.35	88.97
Nearest neighbour	89.85	88.45	89.15	90.09	88.60	89.34

Table 5.12: Comparison of four different approaches to solving the out-of-vocabulary (OOV) problem with the German custom-trained word embeddings.

Once again, it shows that the zero and random embeddings are pretty close to each other when comparing the F1-scores, this time around the zero embeddings performing slightly better on the development set. For German, the averaging technique produces better results than the previously mentioned approach but still does not surpass fastText-generated embeddings, which improve performance with 0.5% on the development set compared to the original zero embedding version. The statistics of the OOV word predictions can be found in Table 5.13. In the case of the average approach, even though there is a clear increase in the F1-score achieved on the development set, there is barely an increase in the number of correctly predicted OOV entities. But the system using the nearest neighbour method clearly benefits from the dedicated embeddings, as the number of correctly predicted entities increased on both development and test sets, thus clearly improving both F1-scores.

German OOV approach	Development set			Test set		
	OOV named entities	Correct %	Incorrect %	OOV named entities	Correct %	Incorrect %
Zero	629	63.59	36.41	663	64.56	35.44
Random	629	63.75	36.25	663	65.16	34.84
Average	629	65.98	34.02	663	64.86	35.14
Nearest neighbour	0	79.01	20.99	0	80.69	19.31

Table 5.13: Statistics on the prediction of the OOV named entities in the German test and development sets. While the nearest neighbour approach essentially creates embeddings for all the OOV named entities, the same entities are checked to see if the dedicated embeddings helped to predict them correctly.

5.5 Experiments with the Hungarian language

The final language in which the experiments were conducted was Hungarian. Once again, four different LSTM hidden dimensions and batch sizes were tested to see which one performs the best. Unfortunately, pretrained GloVe embeddings for Hungarian are not available, so instead a pretrained embedding trained with a combination of Word2vec and fastText made by Szántó et al. [35] was used. The best results can be found in Table 5.14.

Hungarian embeddings	Trained on	Development set			Test set		
		Precision	Recall	F1-Score	Precision	Recall	F1-Score
Pretrained fastText	Custom	92.05	91.84	91.95	91.72	91.23	91.47
	Wikipedia	92.21	92.13	92.17	92.46	92.03	92.25
Custom-trained fastText	Custom	92.55	92.23	92.39	92.63	92.25	92.44

Table 5.14: Comparison of the model's performance on the WikiAnn Hungarian dataset with three different word embeddings. All of these experiments were carried out with an LSTM hidden dimension of 200 and a batch size of 32. Custom-trained embeddings were trained on the corpora found in Table 3.5.

The results show that the custom-trained embeddings are slightly superior to the other two pretrained variants. For the Hungarian language, subword information is crucial, as it adds various affixes to the stem of a word to create new ones. Exactly because of this property, Hungarian word embeddings contain a high percentage of OOV words, even though a lot of words actually originate from the same stem, but even if they differ in one letter, an entirely new embedding would have to be added. This is exactly why solving the OOV problem for this language is of utmost importance. The number of OOV words can be found in Table 5.15

Hungarian embeddings	Trained on	Unique words	WikiAnn Hungarian dataset			
			Perfect match	Case match	OOV words	OOV %
Pretrained fastText	Custom	2 618 006	46 012	522	14 168	23.34
	Wikipedia	793 630	24 801	24 932	10 969	18.07
Custom-trained fastText	Custom	1 406 719	48 732	458	11 512	18.96

Table 5.15: The out-of-vocabulary (OOV) statistics of the different Hungarian embeddings. While the pretrained custom dataset has more than three times the unique words, it still performs worse than the one trained on Wikipedia data, which is due to the high percentage of OOV words.

From these statistics, it is certain that having the least amount of OOV words does not immediately guarantee the highest performance, yet having an overall lower percentage of OOV words definitely has a noticeable positive effect. Once again, the objective is to test the different approaches for removing the OOV words and see if there is a notable improvement. The best performance of the four different approaches can be found in Table 5.16.

Hungarian OOV approach	Development set			Test set		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Zero	92.55	92.23	92.39	92.63	92.25	92.44
Random	92.35	92.26	92.30	92.34	92.39	92.36
Average	92.42	92.27	92.34	92.39	91.96	92.17
Nearest neighbour	93.54	93.25	93.39	93.28	92.80	93.04

Table 5.16: Comparison of four different approaches to remove or reduce the out-of-vocabulary (OOV) problem with the Hungarian custom-trained word embeddings.

Again, the zero and random embeddings do not have a significant difference in performance, even the averaging technique barely improves the score achieved on the development set. However, the nearest neighbour approach has the biggest impact yet on the development set, improving the original score achieved with the original approach by a whole percent. This is most likely because the custom-trained fastText model encountered the stem of most OOV words, and thus was able to create accurate representations for a large percentage of these never-before-seen words. This is supported by the number of correctly predicted OOV entities found in Table 5.17. Therefore, this method might be extremely effective when applied to languages that use multiple affixes to create new words.

Hungarian OOV approach	Development set			Test set		
	OOV named entities	Correct %	Incorrect %	OOV named entities	Correct %	Incorrect %
Zero	1436	79.74	20.26	1452	80.85	19.15
Random	1436	80.78	19.22	1452	81.82	18.18
Average	1436	80.92	19.08	1452	80.99	19.01
Nearest neighbour	0	89.14	10.86	0	88.5	11.50

Table 5.17: Statistics about the prediction of the OOV named entities in the Hungarian test and development sets. While the nearest neighbour approach essentially creates embeddings for all the OOV named entities, the same entities are checked to see if the dedicated embeddings helped to predict them correctly.

5.6 Comparison with state-of-the-art models

With the experiments completed, the next step is to see how the model performs compared to state-of-the-art technologies. For each language, the model with the highest F1-score is used as the basis for comparison, starting with English. As the CoNLL 2003 dataset is quite popular in the circles of NER, there is a myriad of different approaches existing for the English language. The comparison in Table 5.18 contains models that had a large influence on the development of the NER task.

NER model	Author	Word embeddings	Model trained on	Model training time	CoNLL 2003 English dataset
					Test F1
SENNA	Collobert et al. [8]	50D SENNA	CPU	7 weeks	89.59
BiLSTM + charCNN + lex	Chiu & Nichols [5]	50D SENNA	CPU	6 h	91.26
BiLSTM + CRF + ELMo	Peters et al. [26]	50D SENNA	–	–	92.22
BiLSTM + CRF + Context	Akbik et al. [2]	GloVe	GPU	168 h	93.09
BERT-Large + CMV	Luoma & Pyysalo [16]	Wikipedia + BookCorpus	16 TPU	96 h	93.74
BiLSTM + CRF + ACE	Wang et al. [41]	GloVe + fastText + ...	GPU	45 h	94.60
BiLSTM + charCNN + CRF	Thesis model	GloVe	GPU	2 h	90.98

Table 5.18: Comparison of statistics detailing training and consistently achieved F1-score on the CoNLL 2003 test set by different state-of-the-art models.

The model presented in this thesis used the work of Chiu & Nichols as the basis, and while their work achieved a higher score on the test set, it is important to remember that the goal is always to maximise the score achieved on the development set. Fortunately, the work of Chiu & Nichols states that their model achieved a 94.03%, while the model presented in this thesis reached a 94.82% F1-score on the development set.

As the WikiAnn dataset is fairly new, it is currently not widely used yet for tackling NER related problems for multiple languages. Another problem is that the sizes of the train, development and test sets differ, which can lead to quite different results. This is because the original WikiAnn dataset made by Pan et al.[22] did not contain balanced sets; those were created by Rahimi et al.[27]. Even so, there are some community-created NER systems that can be used as a basis for comparison, just to have a better understanding of how precise the thesis model actually is. With these things in mind, Table 5.19 contains the comparison between German NER systems using the WikiAnn dataset, and Table 5.20 shows the same for the Hungarian language.

NER model	Author	Word embeddings	Model trained on	Model training time	WikiAnn German dataset
					Test F1
Bi-LSTM + CRF	Rahimi et al. [27]	300D fastText	GPU	–	89.00
24-layer Transformer (Zero-shot)	Chung et al. [7]	Wikipedia	64 TPU	16 h	84.00
Bi-LSTM + charCNN + CRF	Thesis model	Custom (Table 3.4)	GPU	2 h	89.34

Table 5.19: Comparison of training statistics and the highest F1-scores achieved on the test set of the WikiAnn German dataset.

NER model	Author	Word embeddings	Model trained on	Model training time	WikiAnn Hungarian dataset
					Test F1
Bi-LSTM + CRF	Rahimi et al. [27]	300D fastText	GPU	–	90.00
Fine-tuned BERT	Taner Akdeniz	Webcorpus 2.0	v3-256 TPU Pod	47 h	94.26
Bi-LSTM + charCNN + CRF	Thesis model	Custom (Table 3.5)	GPU	2h 30m	93.04

Table 5.20: Comparison of training statistics and the highest F1-scores achieved on the test set of the WikiAnn Hungarian dataset. Taner Akdeniz is the creator of the community model[3] which is fine-tuned on the WikiAnn German dataset. Training statistics are taken from the original work of Nemeskey [20, 21]

On the basis of these comparisons, it is clear that there has been a lot of improvement over the years, slowly but surely improving the accuracy of NER systems. Advancements such as Embeddings from Language Models (ELMo) [26], contextual string embeddings [2], Bidirectional Encoder Representations from Transformers (BERT) [9], Contextual Majority Voting (CMV), and Automated Concatenation of Embeddings (ACE) [41] all had a large impact on the performance of NER as a whole. However, this does not mean that Convolutional Neural Networks or Conditional Random Fields have become obsolete; it is more about computational power, as pretraining BERT or ELMo requires much larger computational resources while the approaches explored in this thesis are budget friendly in terms of computation.

In conclusion, although the thesis model performed reasonably well on the German WikiAnn dataset, it still underperformed on the Hungarian and English datasets when compared to state-of-the-art models, meaning that there is substantial room for improvement.

[3]https://huggingface.co/akdeniz27/bert-base-hungarian-cased-ner

Chapter 6

Conclusion

The objective of this thesis was to study and create a baseline system for Named Entity Recognition (NER) based on the works of Chiu & Nichols [5] and Abujabal & Gaspers [1]. With the system ready, the main goal was to see how subword information influences the results produced by the model, not only for the English language but also for two other Indo-European languages as well. With the created system, hundreds of experiments were conducted to see how different neural networks, hyperparameters, and subword features influence the model when working with the English dataset. The system started with using only a bidirectional Long Short-Term Memory network and achieving a mere 87.59% on the testing dataset. With two more modules added, such as the Convolutional Neural Network to extract word embeddings from characters, and a linear-chain Conditional Random Field to create dependencies between words, and the small addition of sparse features, the previous score already shoots up to 90.76%. With all the neural networks present, the next step was to try to optimise the model's hyperparameters, with the aim of improving performance little by little. With all the different hyperparameters tested and finalised, the model already achieved an F1-score of 90.98%, which was the highest during the testing. The experiments continued with the testing of multiple pretrained and one custom-trained word embeddings, where on the latter different approaches were attempted to solve one of the major challenges in NER which is the out-of-vocabulary (OOV) problem, but none of the methods could match the accuracy provided by the pretrained GLoVe embeddings. With a fine-tuned model, two Indo-European languages were chosen, namely German, and Hungarian, and the same experiments were carried out to see how influential subword information is when applied to different languages. On the German language, the custom-trained word embeddings with the nearest neighbour OOV approach far outperformed any pretrained variant, achieving an F1-score of 89.34%. The same happened with the Hungarian language, the custom trained variant using the nearest neighbour approach surpassed every pretrained embedding, finishing at a 93.04% F1-score. Both of these languages attest that subword information is indeed an important factor in NER tasks. For future work, I would propose the usage of a subword-based tokenizer called Byte-Pair Encoding (BPE). It would help solve the OOV problem during the tokenization process, as it represents each common word as one token and concentrates on less frequent words by breaking them down into multiple subword tokens. For example, a rare word like impetus would be represented as substrings 'imp', 'et', and 'us' as they are commonly pairs in more frequent words. Modern NER systems already use an improved version of BPE called WordPiece.

Bibliography

[1] Abujabal, A. and Gaspers, J. Neural Named Entity Recognition from Subword Units. In: *Proc. Interspeech 2019* [online]. September 2019, p. 2663–2667 [cit. 2022-4-1]. DOI: 10.21437/Interspeech.2019-1305.

[2] Akbik, A., Blythe, D. and Vollgraf, R. Contextual String Embeddings for Sequence Labeling. In: *Proceedings of the 27th International Conference on Computational Linguistics* [online]. Santa Fe, New Mexico, USA: Association for Computational Linguistics, August 2018, p. 1638–1649 [cit. 2022-04-06]. Available at: https://aclanthology.org/C18-1139.

[3] Bojanowski, P. et al. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* [online]. Cambridge, MA: MIT Press. 2017, vol. 5, p. 135–146, [cit. 2022-4-1]. DOI: 10.1162/tacl_a_00051.

[4] Brownlee, J. Machine Learning Mastery. *What are word embeddings for text?* [online]. August 2019 [cit. 2022-4-1]. Available at: https://machinelearningmastery.com/what-are-word-embeddings/.

[5] Chiu, J. P. and Nichols, E. Named Entity Recognition with Bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics* [online]. Cambridge, MA: MIT Press. 2016, vol. 4, p. 357–370, [cit. 2022-4-1]. DOI: 10.1162/tacl_a_00104.

[6] Christopher, O. *Understanding LSTM Networks* [online]. August 2015 [cit. 2022-4-1]. Available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[7] Chung, H. W. et al. Improving Multilingual Models with Language-Clustered Vocabularies. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* [online]. Association for Computational Linguistics, November 2020, p. 4536–4546 [cit. 2022-04-06]. DOI: 10.18653/v1/2020.emnlp-main.367.

[8] Collobert, R. et al. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research* [online]. 2011, vol. 12, no. 76, p. 2493–2537, revised 11. 10. 2011, [cit. 2022-4-1]. Available at: http://jmlr.org/papers/v12/collobert11a.html.

[9] Devlin, J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human*

Language Technologies [online]. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, vol. 1, p. 4171–4186 [cit. 2022-04-06]. DOI: 10.18653/v1/N19-1423.

[10] Graves, A. and Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* [online]. 2005, vol. 18, no. 5, p. 602–610, [cit. 2022-4-1]. DOI: https://doi.org/10.1016/j.neunet.2005.06.042. ISSN 0893-6080.

[11] Hochreiter, S. and Schmidhuber, J. Long Short-term Memory. *Neural computation* [online]. december 1997, vol. 9, p. 1735–1780, [cit. 2022-4-1]. DOI: 10.1162/neco.1997.9.8.1735.

[12] Jurafsky, D. and James, H. M. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition* [online]. 3 draftth ed. January 2022 [cit. 2022-4-1]. 164–165 p. Available at: https://web.stanford.edu/~jurafsky/slp3/ed3book_jan122022.pdf.

[13] Kumar, E. *Natural language processing.* IK International Pvt Ltd, 2011. 1-2 p. ISBN 978-93-80578-77-4.

[14] Liddy, E. D. Natural language processing. In: *Encyclopedia of Library and Information Science* [online]. 2nd ed. Marcel Decker, 2001, p. 2–3 [cit. 2022-4-1]. Available at: https://surface.syr.edu/cgi/viewcontent.cgi?article=1043&context=istpub.

[15] Lison, P. and Tiedemann, J. OpenSubtitles2016: Extracting Large Parallel Corpora from Movie and TV Subtitles. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)* [online]. Portorož, Slovenia: European Language Resources Association (ELRA), May 2016, p. 923–929 [cit. 2022-04-06]. Available at: https://aclanthology.org/L16-1147.

[16] Luoma, J. and Pyysalo, S. Exploring Cross-sentence Contexts for Named Entity Recognition with BERT. In: *Proceedings of the 28th International Conference on Computational Linguistics* [online]. Barcelona, Spain (Online): International Committee on Computational Linguistics, December 2020, p. 904–914 [cit. 2022-04-06]. DOI: 10.18653/v1/2020.coling-main.78.

[17] Mansouri, A., Affendey, L. and Mamat, A. Named entity recognition approaches. *International Journal of Computer Science and Network Security* [online]. february 2008, vol. 8, no. 1, p. 339, revised 2.20, 2008, [cit. 2022-4-1]. ISSN 1738-7906. Available at: http://paper.ijcsns.org/07_book/200802/20080246.pdf.

[18] Mikolov, T. et al. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR* [online]. january 2013, p. 1–12, [cit. 2022-4-1]. DOI: 10.48550/ARXIV.1301.3781.

[19] Mohit, B. Named Entity Recognition. In: [online]. March 2014, p. 224–226 [cit. 2022-4-1]. DOI: 10.1007/978-3-642-45358-8_7. ISBN 978-3-642-45357-1.

[20] Nemeskey, D. M. *Natural Language Processing Methods for Language Modeling* [online]. 2020. [cit. 2022-04-06]. Dissertation. Eötvös Loránd University. Available at: https://hlt.bme.hu/en/publ/nemeskey_2020.

[21] Nemeskey, D. M. Introducing huBERT. In: *XVII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2021)* [online]. Szeged: [b.n.], 2021, p. 1–12 [cit. 2022-04-06]. Available at: https://hlt.bme.hu/media/pdf/huBERT.pdf.

[22] Pan, X. et al. Cross-lingual name tagging and linking for 282 languages. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* [online]. Vancouver, Canada: Association for Computational Linguistics, July 2017, vol. 1, p. 1946–1958 [cit. 2022-4-1]. DOI: 10.18653/v1/P17-1178.

[23] Paszke, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems* [online]. Curran Associates, Inc., 2019, vol. 32, p. 1–12 [cit. 2022-04-06]. Available at: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[24] Peng, F. and McCallum, A. Information extraction from research papers using conditional random fields. *Information Processing & Management* [online]. Elsevier BV. july 2006, vol. 42, no. 4, p. 965, [cit. 2022-4-1]. DOI: 10.1016/j.ipm.2005.09.002.

[25] Pennington, J., Socher, R. and Manning, C. GloVe: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* [online]. Doha, Qatar: Association for Computational Linguistics, October 2014, p. 1532–1543 [cit. 2022-4-1]. DOI: 10.3115/v1/D14-1162.

[26] Peters, M. E. et al. Deep Contextualized Word Representations. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* [online]. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, vol. 1, p. 2227–2237 [cit. 2022-04-06]. DOI: 10.18653/v1/N18-1202.

[27] Rahimi, A., Li, Y. and Cohn, T. Massively Multilingual Transfer for NER. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* [online]. Florence, Italy: Association for Computational Linguistics, July 2019, p. 151–164 [cit. 2022-04-06]. DOI: 10.18653/v1/P19-1015.

[28] Ramshaw, L. and Marcus, M. Text Chunking using Transformation-Based Learning. In: *Third Workshop on Very Large Corpora* [online]. 1995, p. 6 [cit. 2022-4-1]. Available at: https://aclanthology.org/W95-0107.

[29] Reimers, N. and Gurevych, I. Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing* [online]. Association for Computational Linguistics, November 2020 [cit. 2022-04-06]. Available at: https://arxiv.org/abs/2004.09813.

[30] Rozis, R. and Skadiņš, R. Tilde MODEL - Multilingual Open Data for EU Languages. In: *Proceedings of the 21st Nordic Conference on Computational Linguistics* [online]. Gothenburg, Sweden: Association for Computational Linguistics, May 2017, p. 263–265 [cit. 2022-4-1]. Available at: https://aclanthology.org/W17-0235.

[31] Sakib, S. et al. An Overview of Convolutional Neural Network: Its Architecture and Applications. [online]. version 4.0. Preprints 2018. november 2018, p. 1, [cit. 2022-4-1]. DOI: 10.20944/preprints201811.0546.v4.

[32] Schwenk, H. et al. WikiMatrix: Mining 135M Parallel Sentences in 1620 Language Pairs from Wikipedia. In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume* [online]. Association for Computational Linguistics, April 2019, p. 1351–1361 [cit. 2022-4-1]. DOI: 10.18653/v1/2021.eacl-main.115.

[33] Sokolova, M., Japkowicz, N. and Szpakowicz, S. Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation. In: *AI 2006: Advances in Artificial Intelligence* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, p. 1015–1021 [cit. 2022-04-06]. DOI: 10.1007/11941439_114. ISBN 978-3-540-49787-5.

[34] Sutton, C. An Introduction to Conditional Random Fields. *Foundations and Trends in Machine Learning* [online]. Now Publishers. 2012, vol. 4, no. 4, p. 268–270, [cit. 2022-4-1]. DOI: 10.1561/2200000013.

[35] Szántó, Z., Veronika, V. and Farkas, R. Magyar nyelvű szó- és karakterszintű szóbeágyazások. In: *Magyar Számítógépes Nyelvészeti Konferencia* [online]. University of Szeged, 2019, vol. 13, p. 323–328 [cit. 2022-04-06]. Available at: http://acta.bibl.u-szeged.hu/59021/1/msznykonf_013_323-328.pdf.

[36] Tiedemann, J. Parallel Data, Tools and Interfaces in OPUS. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)* [online]. Istanbul, Turkey: European Language Resources Association (ELRA), May 2012, p. 2214–2218 [cit. 2022-4-1]. ISBN 978-2-9517408-7-7. Available at: http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

[37] Tjong Kim Sang, E. F. and De Meulder, F. 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* [online]. Edmonton, Canada: Association for Computational Linguistics, 2003, vol. 4, p. 142–147 [cit. 2022-4-1]. DOI: 10.3115/1119176.1119195.

[38] Treviso, M. Towards Data Science. *Implementing a linear-chain conditional random field (CRF) in pytorch* [online]. March 2019 [cit. 2022-4-1]. Available at: https://towardsdatascience.com/implementing-a-linear-chain-conditional-random-field-crf-in-pytorch-16b0b9c4b4ea.

[39] Wang, D. and Nyberg, E. A Long Short-Term Memory Model for Answer Sentence Selection in Question Answering. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* [online]. Beijing, China: Association for Computational Linguistics, July 2015, vol. 2, p. 707–712 [cit. 2022-4-1]. DOI: 10.3115/v1/P15-2116.

[40] Wang, S.-C. Artificial Neural Network. In: *Interdisciplinary Computing in Java Programming* [online]. Boston, MA: Springer US, 2003, p. 81–82 [cit. 2022-4-1]. DOI: 10.1007/978-1-4615-0377-4_5. ISBN 978-1-4615-0377-4.

[41] Wang, X. et al. Automated Concatenation of Embeddings for Structured Prediction. *Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing* [online]. august 2021, p. 1–18, [cit. 2022-04-06]. Available at: https://faculty.sist.shanghaitech.edu.cn/faculty/tukw/acl21ace.pdf.

[42] Yamashita, R. et al. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging* [online]. Springer Science and Business Media LLC. june 2018, vol. 9, no. 4, p. 611–629, [cit. 2022-4-1]. DOI: 10.1007/s13244-018-0639-9.

Appendix A

Content of the submitted storage media

```
/
├── datasets/
│   ├── deu/
│   ├── eng/
│   └── hun/
├── embeddings/
├── results/
├── thesis/
├── character_cnn.py
├── conlleval.py
├── crf.py
├── dataset.py
├── LICENCE
├── model.py
├── ner.py
├── README.md
├── shrink_embeddings.py
├── sparse.py
├── thesis.pdf
├── train.py
├── utilities.py
└── vocabularies.py
```

The README.md file contains all the information necessary to recreate any of the experiments seen in the thesis. The directories found in the storage media briefly contain the following:

1. `datasets/` – contains datasets for each different language, each in its own subfolder named based on the ISO 639-2/T: standard. Each subfolder contains 3 files with the name of the language code, but each set is marked with a different extension: `.train` for the training set, `.testa` for the development set, and `.testb` for the test set

2. `embeddings` – contains the custom-trained embeddings for each language, and the best performing pre-trained GLoVe embedding for the English language

3. `thesis` – contains the LaTeXsource files for creating the thesis
4. `results` – contains the Figures and statistics of the best performing models found in the thesis
5. `README.md` – contains a brief description of each Python script and instructions manual for running the experiments