FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Deep Learning Approach to Named Entity Recognition in Portuguese Texts

Ivo André Domingues Fernandes

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Eugénio Oliveira

Second Supervisor: Henrique Lopes Cardoso

July 27, 2018

A Deep Learning Approach to Named Entity Recognition in Portuguese Texts

Ivo André Domingues Fernandes

Mestrado Integrado em Engenharia Informática e Computação

July 27, 2018

Abstract

The problem of named entity recognition (NER) consists in identifying and classifying named entity mentions in free text. Classification is typically based on predefined categories such as people, organizations or locations. NER is a crucial task in the information extraction pipeline and many natural language processing (NLP) tasks depend on the results of NER. NER systems started as rule based engines and later evolved to machine learning approaches such as conditional random fields.

In recent years deep learning approaches have proven to outperform traditional machine learning methods in many natural language processing tasks. Deep Neural Networks are a family of machine learning methods that have been applied to many different fields such as computer vision and speech recognition. The great appeal of deep learning techniques is the fact that the critical process of feature engineering is embedded in the architecture and no longer requires the feature set to be predefined.

Up to this moment there is few research in using deep learning applied to NER in Portuguese texts. This work exposes some of the challenges and limitations of applying a deep learning architectures to NER for the Portuguese language.

This work was split into several stages, starting with analysis and preparation of the textual corpora, followed by defining an evaluation method that will be used to test and compare the different models created. And finally studying, implementing and testing multiple deep learning architectures.

The textual corpora used in this work is split into two categories: annotated and non annotated datasets. Several annotated datasets were used in various experiments, these include: HAREM I GC, HAREM II GC, MiniHAREM GC and WikiNER. Non annotated data was also a crucial part of this work as it is the base for the bootstrapping process, there were two sources of raw textual data: Wikipedia articles and Portuguese news media articles.

The results obtained did not improve the state of the art for NER in the Portuguese language but helped better understand what the challenges and roadblocks are compared to other more researched languages.

Taking into consideration that this work was done without any linguistics background , the results obtained prove that deep learning architectures are very powerful. All feature engineering was handled by the architecture and not by a linguistics expert, this means the process of creating state of the art NLP systems for the Portuguese language has the potential of being improved as long as there is more research and resources available in the future.

Resumo

O problema de reconhecimento de entidades (RE) consiste em, olhando para texto livre, identificar e classificar entidades. A classificação é feita tendo em conta um conjunto predefinido de classes tais como pessoas, organizações ou locais. Existem abordagens clássicas de *machine learning* para este problema mas nos últimos anos abordagens que usam *deep learning* provaram conseguir ultrapassar os métodos tradicionais para muitas das tarefas de processamento de linguagem natural.

RE é uma etapa crucial no processo de extração de informação e existem várias tarefas de processamento de linguagem natural que dependem de RE.

Deep Neural Networks são uma família de métodos de *machine learning* que são usados com grande sucesso em vários campos de ciência dos computadores como por exemplo o reconhecimento de voz ou a visão por computador. A grande vantagem destas técnicas é o facto de que uma das etapas críticas o *feature engineering* está embebido na própria arquitetura deixando de ser necessário a existência de um *feature set* predefinido.

Até este momento existe pouca investigação no que toca ao uso de *deep learning* aplicado ao RE em textos Portugueses, este trabalho ajuda a expor os desafios, as limitações e a viabilidade de aplicar arquiteturas de *deep learning* ao RE para Português.

O trabalho foi dividido em várias fases, começando por analisar e preparar os dados textuais. De seguida definir um método de avaliação que será usado para testar e comparar os modelos criados. E finalmente estudar, implementar e testar múltiplas arquiteturas de *deep learning*.

Os dados textuais usados neste trabalho estão separados em duas categorias: dados anotados e não anotados. Vários *datasets* anotados são usados nas variadas experiências: *HAREM I GC*, *HAREM II GC*, *MiniHAREM GC* e *WikiNER*. Os dados não anotados, isto é texto limpo, são também uma parte crucial deste trabalho pois são a base para o processo de *bootstrapping*. Duas fontes de dados não anotados foram usadas: artigos da Wikipedia e artigos noticiosos de jornais Portugueses.

Os resultados obtidos não melhoraram o estado da arte de RE para a língua Portuguesa mas ajudaram a perceber quais os desafios e dificuldades que existem em comparação com outras línguas mais estudadas.

Tendo em consideração o facto de este trabalho ser feito sem conhecimento especializado em linguítica, os resultados salientam o poder das arquitetura de *deep learning*. Todo o *feature engineering* é feito pela arquitetura e não por um especialista em línguistica, isto significa que o processo de criação de sistemas que superam o estado da arte para a lingua Portuguesa pode ser melhorado desde que, no futuro, exista mais investigação e mais recursos disponíveis.

Acknowledgements

I would like to express my gratitude to Prof. Eugénio Oliveira and Prof. Henrique Cardoso for guiding me during this work, always being open to discuss problems or ideas and providing motivation to explore even further.

I am also thankful to Pedro Saleiro, who guided me through my first steps in NLP and DL, thank you for finding some time to point me in the right direction and push me to work hard.

I wish to thank Gil Rocha for helping with the data gathering process and providing helpful insights to problems and possible solutions.

E finalmente um agradecimento muito especial aos meus pais e a toda a minha família, não seria possível sem vocês.

Ivo André Domingues Fernandes

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

POS	Part-of-Speech
NER	Named Entity Recognition
NN	Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
LSTM	Long Short Term Memory
MEMM	Maximum Entropy Markov Model
CRF	Conditional Random Field
HMM	Hidden Markov Model
GC	Golden Collection
DL	Deep Learning
GPU	Graphics Processing Unit

Chapter 1

Introduction

The popularization of virtual assistants shed light into the importance of many natural language processing tasks and their usefulness for the everyday life of many people. Writing text messages, setting reminders or calling someone are some of the simple actions that can be done solely using voice commands thanks to the combination of multiple natural language tasks. This is just one of the many practical examples where natural language processing is essential.

1.1 Context

Natural language processing (NLP) is a field of science computer science that incorporates the fields of artificial intelligence, computational linguistics and more. NLP is focused on creating ways for computers to process large natural language corpora. To fully process and understand the content of corpora, scientists and researchers have subdivided this macro problem into smaller subtasks with specific goals. These subtasks include word sense disambiguation, stemming, part of speech tagging, chunking, named entity recognition (NER, the focus of this work), co-reference resolution, sentiment analysis, discourse analysis and more [DM14].

Deep Neural Networks refer to a family of machine learning methods that have been applied to many different computer science fields such as computer vision and natural language processing tasks, namely named entity recognition [SM13]. Deep learning (DL) techniques are very appealing since they tend to avoid hand-crafted features, something necessary for classical machine learning techniques. When provided with enough data, DL methods are capable of identifying relevant features, leading to good performance without any external hand-designed resources or time-intensive feature engineering.

Named entity recognition is a task that aims at identifying and classifying entity mentions in free text; these entities are sometimes referred to as *proper nouns*. If we want to process the Portuguese sentence "O José comprou 100 ações da Apple em Janeiro de 1999." the expected result of the NER task applied to this sentence would be:

O [José]$_{\text{Person}}$ comprou 100 ações da [Apple]$_{\text{Organization}}$ em [Janeiro de 1999]$_{\text{Date}}$

Research in NER applied to the Portuguese language started with the HAREM contest in 2004. This contest received submissions from multiple countries and is recognized as being the first evaluation contest for named entity recognition in Portuguese [SSCV06].

Nowadays, the focus of research in NER is the application of deep learning methods, but this is only true for languages with a big research community such as the English language. Having a big research community means there is more annotated corpora available, which is a requirement for all supervised machine learning algorithms; DL methods, in particular, benefit from corpora with a larger size [GBC16]. Looking back at research done in the last few years on NER applied to Portuguese it is hard to find DL methods, with just one particular instance standing out – the work of Santos and Guimarães [dSG15].

The work of this dissertation focuses on experimenting with multiple DL architectures applied to NER for the Portuguese language in order to evaluate the applicability of DL approaches to NER for Portuguese texts and highlight the challenges when compared to more researched languages such as English. To overcome the limitation of the small amount of annotated data available a bootstrapping approach is implemented where non annotated textual data is used to help train NER models. Results obtained in the various experiments are compared with the work of Teixeira *et.al.* [TSO11] and the work of Santos *et.al.* [dSG15].

1.2 Problem Definition

The named entity recognition task is typically subdivided into two subtasks: named entity *identification* and named entity *classification*. Identification of the named entities means retrieving all the entity tokens from the free text. Named entity classification focuses on assigning a class to each of the identified entities; often, the set of classes is predefined.

The task of NER has a long history with implementations dating back to the 1990s and mostly focusing on the English language. Early systems were based on hand-crafted rule-based engines with rules created by linguistic specialists, a process that would take months [TRM08]. Next came supervised machine learning algorithms, using annotated texts to automatically train a model that performs NER. Further down the line, deep learning models started to be applied to NLP and "have been shown to perform very well on various NLP tasks such as language modelling, POS tagging, named entity recognition, sentiment analysis and paraphrase detection, among others" [SM13, p. 5].

When looking at the study of NER for the Portuguese language, the earliest works are the results of the HAREM contest that started in 2004. HAREM was the first contest for NER in Portuguese and received submissions from various countries, with some of the submissions later developing into stand-alone NER tools [SSCV06, SPC06, Car12].

The named entity recognition task is one of many tasks included on the information extraction pipeline, many times used as a preprocessing step or data preparation step for other natural language processing tasks. Tasks that depend on NER include entity linking [DMR+15, Ste12, RMD13, SWH15, CM12] and relation extraction [dABV13]. This dependency means that having a good NER model is necessary in order to create a model for a NLP tasks that relies on NER.

Creating a NER tool with good performance is quite challenging, in particular when there is no restriction on textual genre. An aspect that has affected the NER task through the years is the impact that textual genre has on the generated models. Solutions are highly dependent on the textual genre and porting the solution to another text domain is a major challenge, with cases where the performance of the system goes down 20% to 40% of precision and recall [Nad07]. For this reason, most models do not generalize to different genres – if a model is trained using news media texts it will perform poorly when applied to other genres, for example social media texts.

Another issue that NER models have trouble overcoming is associated with the age difference between the train and test datasets. Training a NER model with textual data that was produced at a very different time period from the test textual data normally results in a drop in performance [TSO11].

Most methods and tools for NER require some preprocessing of the text corpora. A common preprocessing step is part-of-speech tagging that consists of associating the part of speech tag to each word in the text. Part of speech tags and other preprocessing tasks are used to provide the model with some more information about the text being processed, which allows for more accurate detection and classification of named entities. All natural language processing tasks have some error associated with them and using multiple tasks in the same pipeline makes it so that the quality of the previous tasks affects the quality of the following ones. It is often the case that mistakes in preprocessing tasks are propagated to the NER model and can affect performance [dABV13].

Another typical application of NER is question answering systems. Given a document, the top level questions to answer are *who*, *when*, *where* and *what* [MUSC+13]. The answer to these questions are in most cases named entities that can be found in the analysed text.

1.3 Objective

This work is focused on applying multiple deep learning architectures to named entity recognition on texts in the Portuguese language. A total of 4 different architectures are tested. The performance of the models created will be compared with previous solutions for NER. Bootstrapping experiments are compared with the work of Teixeira *et.al.* [TSO11] and models trained using annotated datasets are compared with the work of Santos *et.al.* [dSG15].

1.4 Research Guidelines

In order to reach the outlined objective the following research guidelines were defined:

- Compare available NER datasets for the Portuguese language with the ones available for the English language;

- Apply state of the art deep learning architectures for NER in the English texts to Portuguese texts;
- Evaluate different deep learning NER architectures with Portuguese data;
- Explore NER in different Portuguese textual genres;
- Highlight challenges of using deep learning architectures.

During the development of this work some roadblocks were encountered and many lessons were learned. The learned lessons are discussed in section 6.3.

1.5 Dissertation Structure

The rest of this dissertation is organized into five more chapters. In Chapter 2, the state of the art is described and related work is presented. Chapter 3 lists the datasets used and explains the data gathering process. Chapter 4 describes all deep learning architectures used in this work both for NER and training word embeddings and their respective results in the various experiments. Chapter 5 describes the bootstrapping method, all the experiment setups and results obtained. Finally, in chapter 6 objective completion is discussed followed directions for future work.

Chapter 2

State of The Art

The purpose of this chapter is to summarize the various stages involved in named entity recognition, their evolution through time and defining some concepts in more detail. By the end of this chapter it should be clear what is the current state of the art in NER and what makes this work relevant.

2.1 Named Entity

The term *named entity* was first introduced at the 6th Message Understanding Conference (MUC-6) in 1996 [GS96], where the named entity category set included the following classes: PERSON, ORGANIZATION and LOCATION. In addition to the three classes for entity name expressions (nicknamed ENAMEX), the task also involved numerical expressions such as time, money or dates that were nicknamed NUMEX.

To this day, most NER tools keep using these types but with some modifications such as including more classes or creating hierarchical structures of classes. An example of hierarchical named entity types was introduced in [SSN02], this structure contains about 150 named entity types. In the hierarchical structure the type PERSON is a sub-category of the first level category NAME; additionally, numerous new types were added such as COMPANY and SPORTS_TEAM.

The definition of a named entity, however, is still up for debate, as there are many proposed ways to describe and define the meaning of named entity. In [MUSC+13], four definitions are presented and discussed, starting with defining a named entity as a *proper noun*. Proper nouns or common names designate beings and one-of-a-kind items; however, "common characteristics of proper nouns, such as the lack of inflexions or determinants, the lack of lexical meaning and the use of capital letters, are insufficient to describe named entities" [MUSC+13, p. 484].

The second definition consists on saying that a term is a named entity if it classifies as a *rigid designator*. A term is a rigid designator if it refers to the same thing in all possible worlds where the thing exists on; on worlds where the thing does not exits the term has no meaning. However, saying that a term can only be identified as a named entity if it is a rigid designator is too restrictive.

For instance, “President of the United States” is a named entity that represents a person, but the person it represents changes through time, therefore not classifying as a rigid designator.

Another way to define a named entity is the concept of *unique identification*. When first introduced in MUC-6, the named entity task was resumed as: “The expressions to be annotated are ‘unique identifiers’ of entities (organizations, persons, locations), times (dates, times), and quantities (monetary values, percentages)” [GS96]. Marrero *et al.* [MUSC$^+$13] expose a problem with this definition, “we cannot ensure the existence of unique identifiers in all cases because it depends on the objective. For example, the classification of *Airbus A310* as unique identifier of a type of aircraft seems clear, but the classification of the concept *water* does not seem to respond to it being a unique identifier, unless we want to recognize different types of water, such as sparkling water and still water.”[MUSC$^+$13, p. 484].

The final definition of a named entity defended by Marrero *et al.* [MUSC$^+$13] state that what defines a named entity is dependent on the purpose and the domain of application. This can be seen when looking into the categories considered in the different conferences. MUC conferences focused on identifying and classifying people, organizations, locations, times and quantities. All the MUC categories are linked to military application, which is no surprise, as MUC was sponsored by the Defence Advanced Research Projects Agency (DARPA). A typical application case would be to determine the agent name, time, cause and localization of an event. “The definition of NE according to the purpose and domain of application seems the only one consistent with the literature, evaluation forums and tools.” [MUSC$^+$13, p. 484].

2.2 Datasets

In the context of named entity recognition, a dataset is a list of tokens along side with the true named entity labels, these labels identify if a token belongs to an entity and the respective named entity category. Sometimes datasets include other relevant tags; for example in the CoNLL-2003 dataset the part of speech tag and chunk tag are added [SD03].

For the English language a common dataset used to test models and compare results is the CoNLL-2003 dataset, created in the scope of the CoNLL-2003 shared task that was focused on named entity recognition. As for the Portuguese language, the standard dataset used for testing and comparing systems is the HAREM dataset [SC06], the collection used in HAREM contest which was the first advanced NER evaluation contest for the Portuguese language.

Dataset analysis is focused on the reference datasets for English and Portuguese; these are the datasets most researchers resort to when it comes to evaluating their NER systems. However, there are more datasets available specially for the English language but also for the Portuguese language. Some examples of other relevant NER annotated corpora is listed in Table 2.1.

The English subset of the CoNLL-2003 dataset consists of Reuters news stories between August 1996 and August 1997; details of this corpus are summarized in Table 2.2. The CoNLL-2003 dataset uses the three original named entity categories plus one extra category: MISCELLANEOUS. The miscellaneous category is meant for those named entities that do not belong to

	Name	Genre	Tokens
PT	HAREM I GC	Varied	92K
	SIGARRA News Corpus	News articles	–
	WikiNER	Wikipedia articles	3.5M
EN	CoNLL-2003	News Articles	300K
	OntoNotes 5.0	Varied	1.5M
	WikiNER	Wikipedia articles	3.5M

Table 2.1: Relevant NER annotated corpora.

the three original categories (person, location or organization) but still fit into the definition of a named entity. “This includes adjectives, like Italian, and events, like 1000 Lakes Rally, making it a very diverse category.” [SD03, p.143]

Resources are more scarce for the Portuguese language, with the major reference being the collection used in the HAREM contest [SSCV06]. HAREM was the first advanced NER evaluation contest for the Portuguese language and is still viewed as a major reference for NER in Portuguese texts. The collection created was named HAREM’s Golden Collection (GC) and is a combination of texts from several origins and genres. More than half belong to the web genre or the media genre, but some other genres are included, such as E-mail and oral.

Types present in the GC are more elaborate than the ones present in the CoNLL-2003 dataset. A two layer hierarchy is created with the upper level referred to as *category* and the lower layer as *type*. Categories include: person (PESSOA), organization (ORGANIZACAO), time (TEMPO), location (LOCAL), title (OBRA), event (ACONTECIMENTO), abstraction (ABSTRACAO), thing (COISA), value (VALOR), and others (VARIADO). Each category then has a list of types that can be used to classify named entities with finer detail [SSCV06]. GC statistics are available in Table 2.4.

One of the downsides of the GC corpus is the age of textual data used, Teixeira *et.al.* [TSO11] explored the impact of age difference between train and test datasets for NER models and proved that a large age difference between the datasets has a negative impact on the models. The GC corpus is quite old, using texts created as far back as 1997, which can affect performance of models trained using the GC if the test set contains, say, more contemporary texts or the reverse

	Articles	Sentences	Tokens	LOC	MISC	ORG	PER
Training set	946	14987	203621	7140	3438	6321	6600
Development Set	216	3466	51362	1837	922	1341	1842
Test set	231	3684	46435	1668	702	1661	1617

Table 2.2: CoNLL 2003 English data statistics

Token	POS tag	Token tag	NE tag
U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

Table 2.3: CoNLL 2003 dataset notation example

where models trained using recent textual data are tested using the GC corpus.

There are other two datasets related to HAREM, the Mini-HAREM golden corpus and the Second HAREM golden corpus that were created in the following editions of the HAREM contest. Details about all three datasets are further explored in chapter 3.

The SIGARRA News Corpus was created in the scope of the master thesis of André Pires [Pir17] and is composed of news articles collected from the information system of University of Porto, SIGARRA. A total of 905 news articles were collected and manually annotated using eight different named entity categories: Hour, Event, Organization, Course, Person, Location, Date and Organic Unit.

WikiNER is the name for the collection of silver-standard datasets that resulted from the work of Nothman *et.al.* [NRR+13], these datasets are composed of Wikipedia articles that are automatically annotated using the internal structure and categories present in the Wikitext[1] format. All datasets created have a total of 3.5 million tokens and are available in 9 languages: English, German, Spanish, Dutch, Russian, French, Italian, Polish and Portuguese.

OntoNotes [HMP+06, WPR+12] project was a collaborative effort between BBN Technologies, Brandeis University, the University of Colorado, the University of Pennsylvania, and the University of Southern California's Information Sciences with the goal to annotate a large corpus that includes multiple textual genres. The focus was on three languages: English, Chinese

[1]https://www.mediawiki.org/wiki/Wikitext

Size	GC
Words	92761
Text extracts	129
Named entities	5132

Table 2.4: HAREM I Golden Collection statistics

Category	Description
PERSON	People, including fictional
NORP	Nationalities or religious or political groups
FACILITY	Buildings, airports, highways, bridges, etc.
ORGANIZATION	Companies, agencies, institutions, etc.
GPE	Countries, cities, states
LOCATION	Non-GPE locations, mountain ranges, bodies of water
PRODUCT	Vehicles, weapons, foods, etc. (Not services)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK OF ART	Titles of books, songs, etc
LAW	Named documents made into laws
LANGUAGE	Any named language
DATE	Absolute or relative dates or periods
TIME	Times smaller than a day
PERCENT	Percentage (including "%")
MONEY	Monetary values, including unit
QUANTITY	Measurements, as of weight or distance
ORDINAL	"first", "second"
CARDINAL	Numerals that do not fall under another type

Table 2.5: Different named entity categories in the OntoNotes 5.0 dataset. From [WPR$^+$12].

and Arabic with the size of 1.5 million words, 800 thousand word and 300 thousand words respectively. These datasets were created with multiple NLP tasks in mind, NER included. Named entities were classified into 18 different named entity categories (see Table 2.5).

The file structure and notation varies from dataset to dataset. For example, the CoNLL-2003 data format consists in having a word per line with empty lines representing sentence boundaries. Each line contains four fields: the word, part of speech tag, chunk tag and named entity tag [SD03]. The GC follows a different data format, an Extensible Markup Language (XML) styled format that incorporates the named entity tags in the text. The extract `<PESSOA TIPO="INDIVIDUAL" MORF="M,S">Marcelo Calixto </PESSOA>` represents that the words "Marcelo Calixto" are a named entity of category PESSOA and type INDIVIDUAL [SSCV06].

Using the XML styled format, identifying the start and end of the named entity is not a problem, but for the format followed in the CoNLL-2003 dataset it is necessary to have a tag representation scheme to identify the beginning and end of a named entity. A simple IO scheme, inside (I) outside(O), is not sufficient as it is not capable of labelling consecutive named entities. IOB scheme introduces a beginning label, and therefore supports consecutive named entities. The IOB1 variation of the IOB scheme is adopted by the CoNLL-2003 dataset. There are a total of 5 different tag schemes available: IOB1, IOB2, IOE1, IOE2 and IOBES. Differences between these schemes are described in chapter 3.

There is a cost associated with creating new datasets, since manually annotating the texts takes a long time and is prone to errors even when done by specialists. In cases where dataset size and quality is a bottleneck, some techniques such as bootstrapping can be applied.

A good dataset is an invaluable resource, and has two major applications: testing models and training supervised learning models. Aspects to take into consideration when analysing a dataset are: the textual genre, the age of the text, the number of sentences, the number of tokens and also the number of entities for each category.

2.3 Data Preprocessing and Representation

2.3.1 Feature sets

The first thing to consider when trying to create a supervised learning NER model is how to represent the text, more specifically the words. The chosen representation must contain enough information for the model to be able to learn to identity if the word belongs to a named entity or not. One of the ways to represent words in text is using a feature set.

The most simple feature set would be for the model to only have access to the word being processed. If we are processing the Portuguese sentence “O José comprou 100 ações da Apple em Janeiro de 1999.” and want to classify the word “José” the only information our model would have access to would be the current word “José” and nothing else. It is easy to see how this feature set of only the current word would perform badly; even human specialists would struggle in classifying named entities if the only information they had to work with was the word out of its context in the phrase.

In most approaches, feature sets are complex and include features that aim to capture information that the word by itself can not represent. An example of a feature set used in a real system can be seen in Table 2.6.

Word-level features are features that are directly related to word-level information: for example, if a word is capitalized or the word length. On the other hand, document-level or global features are at the level of the document, for example the number of words in the text being processed. Even when using an extensive set of features to describe the current word and including document-level features, there is information that is not captured. To capture context information for a particular word, it is necessary to include features of not only that word but also neighbouring words. To achieve this, windows are used.

To use a window size of two means that the total feature set includes the features of the focus word as well as the features of the two words to the right of the focus word and the features of the two words to the left of the focus word. Meaning a word is represented not only by its features but also by the features of all other words in the window. Teixeira *et.al.* use a window size of 2 together with the feature set described in Table 2.6 to represent words.

Features	Examples
Capitalized word	John
Acronym	NATO
Word length	"musician" - 8
End of sentence	
Syntactic category	"said" - verb
Semantic category	"journalist" - job
Names of people	Barack Obama

Table 2.6: Example of feature set used in real system. Feature set used in the work of Teixeira *et.al.* [TSO11].

2.3.2 Preprocessing tasks

For most NER systems, in order to process the text to extract entities it is first necessary to do some preprocessing steps, which may include splitting the text into tokens, performing part-of-speech tagging and chunking.

Tokenization is the process of given a string of text return a list of tokens that will be used on the following tasks. Usually these tokens are words but the concept of tokenization can be applied in the same way if we want to obtain a list of characters. For example, given the text “That will be a total of $34.55.” and considering the tokens to be words, the result of tokenization is the token list: [’That’, ’will’, ’be’, ’a’, ’total’, ’of’, ’$34.55’, ’.’].

At a first glance, this process does not seem to be very difficult, but there are some special cases that are hard to overcome. Periods and commas have a special meaning when next to numbers, so they should not be split. Emoticons are another special case, they use a range of special symbols and should be interpreted as a single token when processing the text.

Part of speech (POS) tagging is the task of annotating each word in the text with their part of speech tag. It is common for NER systems to require the text input to already have POS tags since the POS tag is part of the feature set.

Chunking (or shallow parsing) is a NLP task that segments the input text into non-overlapping groups of related words and is normally a step done after POS tagging. A simple example to illustrate chunking, consider the phrase: “My dog likes his food.”. The phrase can be split into a total of three chunking groups: “My Dog” as a noun phrase (NP); “likes” as a verb phrase (VP); “his food” as a NP. An example of applying tokenization, POS and chunking to a sentence can be seen in Figure 2.1.

Including chunking information can result in improvements in performance for the subsequent NLP tasks [DM14]; in fact, chunking tags were, along side with the POS tags, one of the tags included in the CoNLL-2003 annotated NER dataset [SD03].

[2]http://www.nltk.org/book/ch07.html

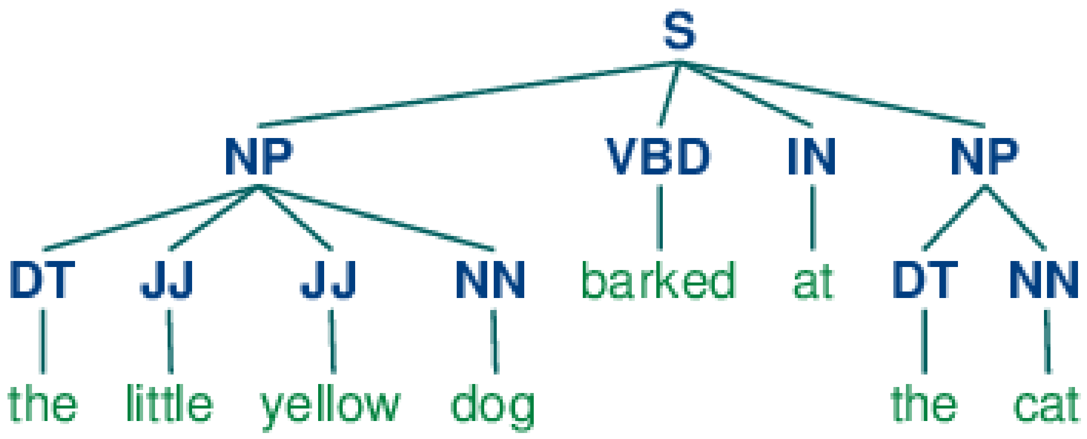

Figure 2.1: Representation of the sentence “the little yellow dog barked at the cat” after tokenization, POS tagging (seen in the leaf nodes) and Chunking (seen at depth 1). From NLTK book[2].

The goal of both stemming and lemmatization is to reduce inflectional forms and related forms of a word to a common base form. For example, considering the list of words *working*, *worker*, and *worked*, all words have a common root word which is *work*.

All tasks presented above can, in some way, be used to help with the NER task either by adding information to the words like POS and stemming or by splitting the text in non trivial ways like chunking and tokenization do.

2.3.3 Word Embeddings

The classical way to represent a word from a vocabulary using a vector is by creating *one-hot vector* representations, where every word is represented by a vector with one position set to the value 1 and all others set to 0. One-hot representations create very sparse vector spaces and the number of dimensions can grow to a huge number depending on the vocabulary size. With a vocabulary size of one million words the number of dimensions would be one million. One-hot word representations do not capture any notion of relationship between similar words. As such, distance between word vectors has no meaning and the vector is simply used as a unique identifier for a specific word.

An alternative to one-hot representations of words is to represent words using *word embeddings* [Mik13, MCCD13]. Word embeddings are distributed representations of words in a vector space and are able to capture linguistic regularities, patterns and even relationships [MCCD13]. With this approach a word is represented as a N dimensional vector, where the vector size N is not dependent on the size of the vocabulary.

With word embeddings, instead of having a very large vector full of zeros and only one cell set to one, the number of dimensions is smaller (typically ranging between 50 to 300) and vector values are real numbers learned using neural network models. Vectorial representations of words were popularized by Mikolov *et.al.* [MCCD13] with the Word2Vec model. They presented two main approaches to learning word embeddings: the SkipGram (SG) approach predicts context words given the centre word; the Continuous Bag of Words (CBOW) approach predicts the centre word given the context words.

Simply put, the SG model receives the centre word one-hot vector as input and outputs context word prediction probabilities that are then compared to the actual context words. Prediction error is calculated and network weights adjusted using back-propagation. At the end of the training phase, network weights in the hidden state contain the word embeddings.

Following the work done by Mikolov *et.al.* new embedding models were developed, including: Wang2Vec [LDBT15], GloVe [PSM14] and FastText [BGJM16]. The GloVe model works with a co-occurrence word matrix, while Wang2Vec, Word2Vec and FastText are composed of predictive methods.

Wang2Vec is a slightly modified version of Word2Vec, modifications were added to take into consideration word order. Word order is important to solve syntax based problems, and Wang2Vec showed improvements in POS and dependency parsing [LDBT15].

FastText approaches the problem in a different way: each word is represented by the sum of vectors representing character n-grams. FastText tries to overcome the problem of obtaining representational vectors for words in languages with large vocabularies and many rare words. Representing each word by a sum of character n-gram representations ensures all words have an unique representational vector meaning *unknown* words for the embedding model are no longer mapped to a vector representation of *unknown* [BGJM16].

A downside of word-level embeddings is that "information about word morphology and shape is normally ignored when learning word representations" [SZ14, p. 1]. There are tasks in NLP such as part-of-speech tagging that require morphological or word shape information; hand-crafted features are used to include this information. Alongside word embeddings, in the work of Namazifar [Nam17] two one-hot vectors are used; one representing the POS tag of the token and another identifying if the token contained special characters.

Character-level representations can capture some information left out by word-level representations. Santos and Zadrozny [SZ14] propose a deep neural network that learns character-level representations of words and in combination with word embeddings creates state-of-the-art POS taggers for both English and Portuguese.

Embeddings are a powerful representation and in most cases have a real chance of improving performance. Passos *et al.* [PKM14] have presented the first system to obtain state-of-the-art results for NER using embeddings. One disadvantage embeddings have is the fact that the time it takes to train can be very long, mainly because huge amounts of textual training data are used. Mikolov *et.al.* [MCCD13] report a training time of two days using 140 CPU cores to train a 1000 dimensional vectors using a 6 billion word vocabulary.

Systems that use embeddings as word representations normally initialize these embeddings with previously trained embeddings. Since most of the final network trainable parameters will be related with word embeddings, initializing them with already good weights gives a head-start to training the network. Examples of this can be seen in [CN15, dSG15, Nam17]. After initializing the network with pre-trained word embeddings the training starts and word embeddings will be fine tuned to adjust to the domain and task at hand. It is also possible to initialize the network with pre-trained embeddings and freeze the embeddings layer, this means word embeddings will not be fine tuned but translates into a decrease in training time.

To evaluate the quality of embeddings there are two options: intrinsic evaluations and extrinsic evaluations. Intrinsic evaluation consists on word vector analogies, essentially doing additions and subtractions with the word vectors and then comparing the result with all words in the vocabulary in order to find the one that is most similar. The measure of similarity is normally the cosine distance between the result and the words in the vector space.

A simple example of one analogy is finding the word in the vector space that relates to the word *small* in the same was as the word *biggest* relates to the word *big*. This analogy is mapped to the following operations: $X = vector(”biggest”) - vector(”big”) + vector(”small”)$. If the word embeddings are well trained, searching for the closest word to X in the vector space will result in finding the word *smallest*, proving that the learned word embeddings successfully capture word relationships. The example used was obtained from the work of Mikolov *et.al.* [MCCD13].

Extrinsic evaluation is more time consuming since it requires the newly trained word embeddings to be tested as part of a system to perform a real task and evaluate the impact the change makes on the performance of the system.

2.4 Methodologies

Early implementations of NER systems were largely based on hand-crafted rules created by specialists, who looked at the features of named entities of different categories and came up with rules to detect and classify them. Features regularly used to categorize named entities include whether the word starts with capital letter, whether it ends with a period and word length. A rule based system can be implemented using, for example, regular expressions.

For research in some languages, such as Portuguese, it took longer for researchers to adopt new methodologies compared to the English language. In 2004, in the first HAREM contest only one of the submissions used a machine learning [SSCV06] approach; in contrast, during the CoNLL-2003 shared task all submissions were machine learning implementations [SD03].

Creating hand-crafted rules is a long process that takes a long time to complete and requires the expertise of linguistics specialists. Machine learning, on the other hand, can obtain models that encode these rules in a fraction of the time. However, supervised machine learning methodologies require annotated examples to train the models, and annotated corpora can in itself be very expensive to obtain.

2.4.1 Classification Methodologies

NER can be viewed as a classification problem: given a word, classify it as a named entity of a specific category or as not a named entity. There are several machine learning classification algorithms that can be applied to the problem, namely support vector machines [TC02, EB10], Naïve Bayes [SS15, AHFB17] and decision trees [AHFB17, SFK06, PKPS00].

2.4.2 Sequential Methodologies

Textual data is in itself sequential. Word order matters and is an essential factor to the meaning of the sentence. NER can be viewed as a sequence labelling problem: given an annotated corpus and a set of features, a sequence classifier is trained to predict the labels from the data.

Both classification and sequential methods require a large enough annotated corpus so that the models can learn what features identify each category. In addition, the feature set needs to be manually chosen and can be highly dependent on context and domain of the datasets used.

Sequential machine learning algorithms include *Hidden Markov Model* [TRM08, ZS01], *Maximum Entropy Markov Model* [BON03, CN03] and *Conditional Random Fields* [TSO11, LZWZ11, dAV13, SZH17, CF13].

A *Hidden Markov Model (HMM)* is a finite state automaton that models a probabilistic generative process where a sequence of observations is produced by starting in some state, emitting an observation selected by the state, changing to another state and repeating the process until a desired final state is reached [MFP00].

With a more formal notation, a HMM is represented by a finite set of states S, a set of observations O, two conditional probability distributions – a state transition probability $P(s|s')$ representing the probability of the current state s given that the previous state is s', with $s, s' \in S$ and an observation probability $P(o|s)$, $o \in O$, $s \in S$ – and finally a initial state distribution $P_0(s)$.

For a natural language processing task such as NER, the observations are typically a vocabulary and sequences of observations are associated with a sequence of labels. Models can associate one or more states to each label. So, given a set of observations, the model can output the most likely label sequence obtained using the path returned by the Viterbi algorithm [For73], that finds the most likely sequence of hidden states.

A *Maximum Entropy Markov Model (MEMM)*, also known as Conditional Markov Model (CMM), is a model based on HMM but "allows observations to be represented as arbitrary overlap of features and defines a conditional probability of state sequences given observation sequences" [MFP00, p. 591].

Comparing MEMM to HMM, we can observe that the two conditional probabilities are replaced by a single conditional probability distribution $P(s|s', o)$, the probability of the current state s given the previous state s' and the current observation o. Unlike HMM where the current observation only depends on the current state, for MEMM the current observation may also depend on the previous state [MFP00].

According to Lafferty *et.al.* [LMP01], *conditional random fields* offer several advantages over both HMMs and MEMMs. MEMMs and other non-generative finite-state models based on next-state models share a weakness that Lafferty *et.al.* called the *label bias problem*.

The label bias problem appears due to the fact that transitions leaving a given state compete only against each other, rather than against all other transitions in the model. In other words, this is a conditional probability of the next state given the current state and the observation sequence. All the mass that arrives at a state must be distributed among the possible successor states. An observation can affect which destination states get the mass but not how much total mass to pass on. This causes a bias towards states with fewer outgoing transitions.

Conditional random fields is a sequence labelling framework that has all the advantages of MEMM but solves the *label bias problem*. The critical difference is that a MEMM uses per-state exponential models for the conditional probabilities of next states given the current state, while CRF has a single exponential model for the joint probability of the entire sequence of labels given the observation sequence.

2.4.3 Deep Learning Methodologies

Aside from the performance improvement, the great appeal of approaches based on deep learning (DL) is the possibility of discarding the need of feature engineering [CN15]. The deep learning model is capable of learning what features to detect and using these features to identify and classify the named entities into the predefined categories.

Deep learning can be described as allowing computers to learn from experience and to represent concepts as a hierarchy where each concept is defined through its relation to simpler concepts. As Goodfellow, Bengio and Courville put it, "If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning." [GBC16, p. 1-2]

Abstract and formal tasks such as playing chess, despite having many possible moves, are still very restricted and follow a strict set of rules. These types of tasks are challenging for humans but technically simple for computers. On the other hand, there are tasks that seem effortless for humans, like speech recognition, but that computers have struggled with for many years.

Handling tasks with specific rules and restrictions is not much of a challenge but the everyday life encompasses tasks that require a large amount of knowledge about the world. This knowledge is subjective and differently interpreted by different people making it difficult to articulate in a formal way interpretable by a machine [GBC16].

Advances in hardware, specially GPU's, have allowed for faster training times and consequently for deep learning solutions to become viable. Having the possibility of training deep learning models in less time and with fewer hardware resources allowed for much more research to be done that resulted in deep learning solutions improving the state of the art of problems in various computer science areas.

The modern deep learning framework is quite versatile. Adding more layers and units within a layer enables a deep learning network to represent functions of great complexity. Tasks that consist of mapping an input vector to an output vector and are intuitive for a person to do rapidly, can be accomplished via deep learning. It is just a matter of having a large enough model and sufficiently large datasets of labelled training examples [GBC16].

The versatility and power of deep learning architectures combined with advancements in hardware pushed DL to be used in many computer science areas, including NLP and more specifically NER. The last few years of research in the English language has mainly focused on DL approaches [LBS+16, SVL14, YZD17, dSG15, CN15, Nam17, CWB+11, CW08, HXY15, SM13]

Feedforward neural networks, or multilayer perceptrons (MLPs), are the standard model for deep learning, and all other models can be seen as variations of this simple architecture. The goal of feedforward neural networks is to approximate some function f^*. “For example, for a classifier, $y = f^*(x)$ maps an input x to a category y. A feedforward network defines a mapping $y = f(x;\theta)$ and learns the value of the parameters θ that result in the best function approximation.” [GBC16, p. 164]

As the name suggests, the information is only fed forward through the network until a result comes out. If the outputs of the model are fed back into itself, a feedback connection is created, resulting on what is called a *recurrent neural network*. The name of these networks also includes *neural* since “each unit resembles a neuron in the sense that it receives input from many other units and computes its own activation value. The idea of using many layers of vector-valued representation is drawn from neuroscience” [GBC16, p. 165]. Feedforward neural networks are named networks because they are a composition of multiple simple functions arranged in what can be described as a directed acyclic graph.

Convolution neural networks (CNNs) are a special kind of neural network for processing data that has a known grid-like topology, such as time-series data or image data. “The name "convolutional neural network” indicates that the network employs a mathematical operation called *convolution*. A convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.” [GBC16, p. 326]

CNNs have had major successes in practical applications. A task that seems to fit perfectly as a use case for CNNs is image processing, as images have a representation that is inherently represented as a 2 dimensional matrix. CNNs can also be applied to NLP, for Santos *et.al.* [dSG15] use a convolutional layer to extract character level representations of words.

Analysing a famous CNN for image classification [KSH12] three different types of layers are identified: convolutional layers, polling layers and dense or fully-connected layers. A convolution can be seen as going through the input data and applying an operation to multiple subsets of the input; the operation is named a kernel or filter. The kernel defines what action to take for the input. Normally kernels are small in terms of width and height but extend through the full depth of the input. In the case where the input to a CNN is an image with dimensions [256,256,3] a typical kernel would be [5,5,3] (5 in width 5 in height and 3 in depth, the colour channels). During the forward pass the kernel convolves across the width and height of the input and dot products between the filter and input positions are computed. In the example of image processing it is expected that some of the learned kernels activate when they see some kind of visual feature, such as an edge or a corner.

Along side with kernel dimensions there are some other hyper-parameters that need to be

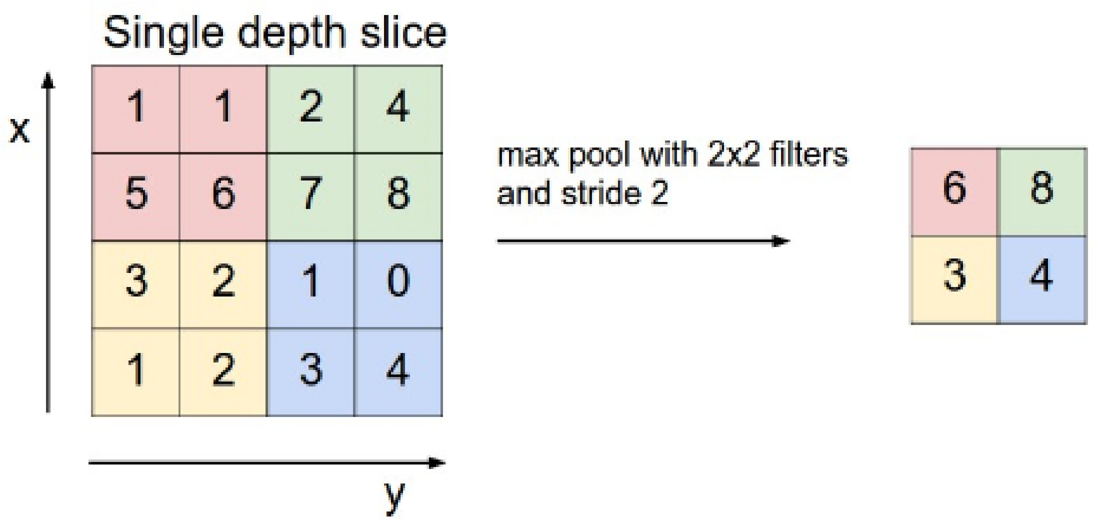

Figure 2.2: Max polling layer in effect. From cs231n website[3].

configured, namely the use of zero-padding and the stride value. Zero-padding means padding the input with zeros around the border, this allows control of the size of the output and is normally used so that output dimension is equal to input dimension. The stride value is related to how many pixels to slide the kernel after each application, when stride is 1 the kernel is moved one pixel at a time, larger stride values mean outputs with smaller dimensions are produced.

Polling layers are commonly used in-between successive convolutional layers, they are used to reduce the dimension of the representation and therefore reducing the amount of parameters and computation required. Polling layers are also responsible for controlling overfitting. The most common polling operation is max polling where the output is the max value for each kernel pass in the input. A simple example of the effect of a max polling layer is illustrated in Figure 2.2. Polling layer can perform other functions, such as average polling or L2-norm polling but the most commonly used is max polling.

Fully-connected or dense layers are connected to all activations in the previous layer and are normally used at the end of the CNN architecture to map the internal representation to the output vector. In the work of Krizhevsky *et.al.* [KSH12] the tasks was to classify an image into 1000 different categories so the last layer of the network is a dense layer that maps the internal representation to a 1000 dimensional vector that can be interpreted as the scores of each of the 1000 different categories.

Recurrent neural networks (RNNs) are a family of neural networks that work on sequential data such as text. Typically, approaches are limited in sequence size, which represents the number of previous inputs that are used in predicting the label for the current input. Increasing the sequence size leads to the memory requirements growing to a point where it is computationally inviable to train the network. RNNs use the output of the previous time step as an input for the next

[3]http://cs231n.github.io/convolutional-networks/#pool

step, essentially allowing the following prediction to be influenced by all previous inputs and predictions.

Infinite sequence length would be ideal since the prediction of the following sequence would be based on the whole previous sequence and not only in a portion of it. In reality having an infinite sequence length is not only limited computationally but also by the vanishing gradient problem and exploding gradient problem. When the sequence length is too large, gradients of the initial sequence inputs either get too close to 0 or explode to infinity meaning no actual learning is done. Even using techniques to counter the problem, like gradient clipping, it is never truly solved.

In the previous explanation the sequence in which the next tag prediction is based on only includes the words up to the focus word, but the context of the word is also affected by the words that follow it. The bidirectional RNN architecture is composed of two RNNs, a forward RNN that processes the input in the normal order and a backward RNN that processes the input in the reverse order. The tag prediction is based on the combined output of both RNNs which means information of both the previous sequence of words and following sequence of words is considered when making the tag prediction.

Long short term memory networks (LSTMs) are a specific case of RNNs, and as the name indicates they are designed to learn long term dependencies, longer than RNNs do. The main components of a LSTM are: a memory cell or cell state, input gate, output gate and forget gate.

At each step, the first decision that needs to be made is to decide what information is no longer relevant and can be thrown away. This is done using the forget gate; irrelevant information is then removed from the cell state. Next is to decided what new information to add to the new cell state. Using the input gate, a decision is made as to what values of the cell state to update, and finally the output gate is used to decide what is exposed by the cell.

The way LSTMs can maintain longer dependencies than RNNs is by making use of gates to selectively keep interesting dependencies and throw away those which have no value.

RNNs, specifically LSTMs, are the architectures most used for NER in the last few years [Nam17, LBS+16, SVL14, YZD17, ABP+16, MH16, HXY15, CN15]. NER can be viewed as a sequence to sequence problem, where the input is a sequence of words and the output is a sequence of named entity tags that match the input words. This concept perfectly fits the RNN architecture. CNNs are also used but not as a main architecture. There are several works that explore the use of CNNs to extract character level information from words [MH16, CN15, dSG15].

Just like classical machine learning architectures, DL network architectures also suffer from the over-fitting problem. One of the ways DL researchers found to deal with this problem is to apply dropout [SHK+14] in some of the layers of the network. Applying dropout means randomly ignoring units from the neural network during training, this prevents units from fitting too strictly to the training data. An example of the application of dropout to a neural network is illustrated in Figure 2.3

Training a neural network means iteratively tweak the network parameters to gradually obtain slightly lower and lower errors over the training dataset. To update each network parameter it is necessary to know how it affect the error function. If the parameter value positively affects

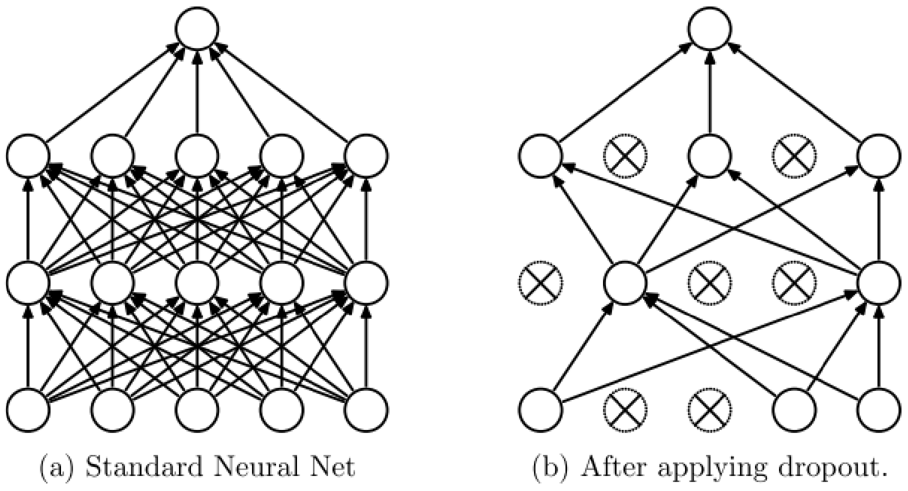

Figure 2.3: Effect of dropout on a neural network. From [SHK+14]

the error function the objective should be to lower the parameter value, on the other hand if the parameter value negatively affects the error function the objective should be to increase it's value. To know exactly how the network parameters affect the error function the gradient of the loss function is calculated.

Now that the gradient for each network parameters is known, finishing the network training iteration is only a matter of updating the parameter value based on the gradient. There are various algorithms that describe exactly how these updates should be made. Gradient descent is one of the most used algorithms to perform neural network optimization [Rud16].

Gradient descent has three variations, differences between them are related to how many training examples are processed before gradients are calculated and network parameters adjusted.

Batch gradient descent requires the gradients for the whole training dataset to be calculated to perform just one update to the network parameters, batch gradient descent can be very slow and becomes unusable if the training dataset does not fit into memory. Stochastic gradient descent (SGD), on the other hand, performs a parameter update for each training example, these frequent updates with high variance cause the objective function to fluctuate heavily. Mini-batch gradient descent combines batch gradient descent and SGD, it performs an update for every mini-batch of n training examples.

In all variations of the gradient descent algorithm the update to the parameters is done using the following rule: $params = params - learning\_rate * params\_grad$. Choosing the proper learning rate can be difficult, too small and the convergence is very slow, too large and it can overshoot the minimum and start diverging. Another big challenge when minimizing highly non-convex functions is avoiding getting trapped in the numerous suboptimal local minima. A number of algorithms and modifications were developed in order to deal with these challenges: Momentum, Nesterov accelerated gradient, Adagrad, Adadelta, RMSprop, Adam, AdaMax, Nadam and more [Rud16]. However no meaningful improvements were observed when using sophisticated optimization algorithms on DL network architectures target at the NER task [CN15, MH16].

2.5 Evaluation Metrics

There are several evaluation metrics used to evaluate performance of NER systems. The major difference between methods is related to how each one defines what an error is and what impact different types of errors have in the performance measure.

Consider the following test case:

O [José]$_{\text{Person}}$ comprou 100 ações da [Apple]$_{\text{Organization}}$ em [Janeiro de 1999]$_{\text{Date}}$

Suppose our system outputs the following named entities:

O [José]$_{\text{Person}}$ comprou 100 ações da [Apple]$_{\text{Miscellaneous}}$ em [Janeiro]$_{\text{Date}}$ de 1999

Only one fully correct named entity is identified, “José”, as a named entity of the class *Person*. But there are other types of measures that take into account multiple types of errors.

The performance of NER systems is measured using precision (Eq. 2.1), recall (Eq. 2.2) and F1-score (Eq. 2.4), which is the F-measure (Eq. 2.3) where both precision and recall have the same weight ($\beta = 1$) for the final score.

$$Precision = \frac{Number\ correct\ NEs\ detected}{Total\ number\ NEs\ detected} \tag{2.1}$$

$$Recall = \frac{Number\ correct\ NEs\ detected}{Total\ number\ NEs\ in\ test\ set} \tag{2.2}$$

$$F_\beta = (1+\beta^2) \cdot \frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall} \tag{2.3}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.4}$$

There are different scoring techniques that basically differ in how they consider the named entity identification and classification to be correct. When taking into consideration identification and classification of named entities, the system can make five different types of errors, as shown in Table 2.7.

Considering these types of errors, there are three main scoring techniques: MUC evaluations, Exact-match evaluations and ACE evaluations.

2.5.1 MUC Evaluations

The performance measure used in the message understanding conference (MUC) evaluates the system on two aspects: ability to classify the named entity into the correct category (TYPE) and ability to identify correctly the named entity on the text (TEXT). The correct category score is

Correct solution	System prediction	Error Type
comprou 100 ações da	comprou 100 [ações]$_{\text{Person}}$ da	Identifies named entity where there is none.
O [José]$_{\text{Person}}$	O José	Misses the named entity completely.
ações da [Apple]$_{\text{Organization}}$	ações da [Apple]$_{\text{Object}}$	Identified the entity but paired it with the wrong label.
ações da [Apple]$_{\text{Organization}}$	ações [da Apple]$_{\text{Organization}}$	Classified the entity correctly but boundaries were incorrect.
ações da [Apple]$_{\text{Organization}}$	ações [da Apple]$_{\text{Object}}$	Wrong boundaries and wrong label.

Table 2.7: Different types of errors.

assigned if a named entity is matched with the correct class ignoring the boundaries as long as there is an overlap. Correct identification score is assigned if the named entity boundaries are correct regardless of the class prediction.

For both types TYPE and TEXT, the three counters necessary to calculate the F-measure are kept: number of correct guesses, number of total guesses and total number of entities on the test set. Analysis of the errors in the example above is presented in Table 2.8 and the respective precision, recall and F1 measures in equations 2.5, 2.6 and 2.7 respectively.

Counter	TYPE	TEXT	Total
Correctly detected	2	2	4
Total detected	3	3	6
Total on test set	3	3	6

Table 2.8: Example counters for MUC evaluation

$$Precision = \frac{4}{6} \tag{2.5}$$

$$Recall = \frac{4}{6} \tag{2.6}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \simeq 0.67 \tag{2.7}$$

This measure has the advantage of taking into account all possible types of errors of Table 2.7. It also gives partial credit for errors occurring on one axis only [Nad07].

2.5.2 Exact-match Evaluations

The performance measure used to evaluate NER systems in the IREX and CONLL conferences is as simple as only considering a predicted named entity correct if it is an exact match both in identification and classification. Comparison between systems is achieved using micro-averaged F-measure with the precision being the percentage of named entities found by the system that are correct and the recall being the percentage of named entities present in the solution that are found by the system [Nad07].

In this metric all the five types of errors presented above in Table 2.7 are interpreted as having the same value, that is, all are considered errors with the same weight.

Looking at the example above, only one named entity is matched correctly which means a lower F1 measure compared to MUC evaluations. Precision, recall and F1 calculations for the exact match performance measure are present in equations 2.8, 2.9 and 2.10 respectively.

$$Precision = \frac{1}{3} \tag{2.8}$$

$$Recall = \frac{1}{3} \tag{2.9}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \simeq 0.34 \tag{2.10}$$

2.5.3 ACE Evaluations

The evaluation used on the ACE conference has a complex structure and design. Each named entity type has a parametrized weight and contributes up to a maximal proportion of the final score (e.g., if each person is worth 1 point and each organization is worth 0.5 point then it takes two organizations to counterbalance one person in the final score). Customizable costs are used for false alarms, missed entities and type errors.

ACE evaluation may be the most powerful evaluation scheme because of its customizable cost of error and its wide coverage of the problem. It is however problematic because the final scores are only comparable when parameters are fixed. In addition, complex methods are not intuitive and make error analysis difficult [Nad07].

2.6 Related Work in Portuguese

As mentioned before, the HAREM contest that took place in 2004 was the first time NER was tackled for the Portuguese language. A second HAREM contest took place in 2008 with some slight changes to the categories and types and a new evaluation measure [FMS+10].

From all the participations on the second HAREM contest, only one adopted a machine learning approach; all others relied on hand-crafted rules in combination with dictionaries, gazetteers and ontologies. “This shows that the community dedicated to NER in Portuguese hasn’t embraced machine learning techniques, contrary to the situation for English.” [FMS+10, p. 3635]

This observation is no longer true, as there is active investigation using machine learning techniques for the Portuguese language [TSO11, NRR+13, GG15, dAV13, VR08, MD07, Sar06].

However, nowadays we notice another trend: research in NLP, specially for English, is focusing on deep learning architectures [SM13, SVL14, YZD17, CWB+11, HXY15, CN15, GBVS15, Nam17]. But looking at the Portuguese language most research still focuses on classical machine learning approaches, an exception is the work of Santos *et.al.* [dSG15] which is the most prominent research on NER applied to the Portuguese language that uses a deep learning approach.

The process described by Teixeira *et.al.* [TSO11] presents a way to train semi-supervised NER models using non-annotated corpora. Creating annotated corpora is difficult and expensive and as a result the available NER-annotated corpora is usually small and composed of texts that are several years old. Unannotated corpora is much more accessible and readily available and can be exploited to create more up to date NER models.

The corpus used by Teixeira *et.al.* [TSO11] consists of 50,000 news items extracted from Portuguese online newspapers between April 2011 and May 2011. Each item contains a title and a body. The number of sentences present is approximately 400,000. The method consists in starting with non-annotated news items and annotate names using a dictionary approach. Next, this corpus is used to infer a CRF model. Using the newly learned model the initial corpus is re-annotated and used once more to train a CRF model. This cycle continues until performance stabilizes. At each step, model performance is measured in different ways: using the HAREM GC, manually evaluating precision and recall over a small sample of news corpus from the original corpus, and checking the correctness and number of new names identified. The results obtained by Teixeira *et.al.* [TSO11] mean that it is a viable option to include non-annotated corpora in the process of creating models for named entity recognition for Portuguese.

Another solution based on traditional sequence labelling machine learning techniques is introduced by Amaral *et.al.* [dAV13]. NERP-CRF is a NER system based on CRF for the Portuguese language. The data used was HAREM corpus. Two experiments were done: one using the golden corpus of the second HAREM to both train and test, and another using the GC of the first HAREM to train and the GC of the second HAREM to test.

Santos *et.al.* [dSG15] use both word embeddings and character embeddings in combination with the neural network architecture for sequential classification presented by Collobert *et.al.* [CWB+11] to create a solution for NER in Portuguese texts. This solution outperformed the state-of-the-art system in the HAREM corpus. The fact that this solution performed so well is a testament to the fact that deep neural networks are suitable for NER in the Portuguese language.

Garcia *et.al.* [GG15] introduced CitiusTools, a multilingual suite for NLP that performs multiple tasks, including NER. The tool started only supporting the Spanish language but in 2015 the tool was extended to support both Portuguese and English. This tool uses a rule based approach to NER, making use of gazetteers, dictionaries and a list of the most frequent personal names for each language.

Another tool that supports the Portuguese language in many NLP tasks is FreeLing [PS12]. This open-source tool started supporting 3 languages (English, Spanish and Catalan) and now supports 9 (added Galician, Italian, Welsh, Portuguese, Asturian and Russian). Both modules involved in NER, recognizer and classifier, are based on the CoNLL-2002 shared task winning system [CMP02]. This machine learning based system makes use of binary AdaBoost classifiers.

spaCy[4] offers a multi-task convolutional neural network that supports part-of-speech tagging, dependency parsing and named entity recognition for the Portuguese language. The spaCy library also includes language models for English, German, Spanish, French, Italian and Dutch.

Natural Language Toolkit (NLTK) is a platform built to work with human language data, provides corpora loaders and trained models for many NLP tasks. Useful models available for the Portuguese language[5] include: word tokenizer, sentence tokenizer, POS tagging and stemming.

2.7 Related Work in English

Maintaining the trend associated with other natural language processing tasks, the English language is the one language that most named entity recognition research focuses on. The work on named entity recognition started in the Message Understanding Conference (MUC-6) in 1996, where the task of NER was first introduced. Further work was done in CoNLL-2002, CoNLL-2003, MUC-7 and continues today.

Collobert *et.al.* [CWB$^+$11] present a neural network architecture. This architecture can be applied to multiple natural language processing tasks. By avoiding task-specific engineering and ignoring man-made input features, the system learns what are the optimal representations and therefore can adapt to multiple tasks. Four NLP tasks are explored: POS tagging, Chunking, NER and semantic role labelling. For the NER task the corpus used is the one made available in the CoNLL-2003, which consists of Reuters news articles, a training set of around 200,000 tokens and test set of approximately 46,000 tokens. Since most of the trainable parameters of the network are related to word embeddings, Collobert *et.al.* initializes the word embedding weights with pre-trained embeddings in order to jump start the learning process. Word embeddings were trained using two datasets: English Wikipedia with 631 million words and Reuters news articles containing 221 million words. The use of pre-trained word embeddings to initialize the word embeddings layer improved performance in all the tasks tested. Collobert *et.al.* [CWB$^+$11] demonstrated that it is possible to create a neural network architecture capable of surpassing state of the art solutions without requiring any external feature engineering or external resources. The flexibility of the network to adapt to different NLP tasks also shows the strength of deep learning approaches when applied to NLP.

Yang *et.al.* [YZD17] present a way to improve baseline NER solutions. Given the output of the baseline solution, all entities are substituted with their class, resulting in sentence patterns such as "PERSON was born in LOCATION". These patterns are then used in training LSTM and CNN structures that will improve the baseline performance. Improvements are based on changing the class of entities when the pattern assigned does not seem to be correct. In the example “U.S. beat El Salvador 3-1” the pattern present is “LOCATION beat LOCATION”. If the baseline

[4]https://spacy.io/models/pt

[5]http://www.nltk.org/howto/portuguese_en.html

incorrectly tags one of the named entities as something other than LOCATION the output can be corrected. The system identified that “LOCATION beat LOCATION” is more likely to be correct than “LOCATION beat ORGANIZATION” or any other combination.

Namazifar [Nam17] set to solve the problem of assigning confidence value to detected named entities. This problem is defined as Named Entity Sequence Classification (NESC). The use case in which including confidence levels is important is for example content recommendation systems. The work focuses on twitter data (tweets). “It is important to note that for recommending content it is crucial to have very high confidence in the detected named entities to be actually true named entities.” [Nam17, p.2]. To achieve this, an architecture based on the one presented by Huang *et.al.* [HXY15] was implemented; 200 dimensional word vectors are trained using Glove [PSM14] on over 1 billion tweets. Along side with word embeddings they use 2 one-hot vectors: one encodes information about word capitalization and presence of special characters; the other vector specifies the part-of-speech tag associated with the token. The full system includes: word embeddings followed by a bidirectional recurrent neural network (bidirectional LSTM) and a fully connected layer followed by a softmax layer that produces a discrete probability distribution for the possible labels. In the last step, probabilities go through a CRF which learns the correct order of entity labels. The dataset used contains 100,000 manually annotated tweets and the training set included 62,507 named entities.

Lample *et.al.* [LBS$^+$16] introduce an architecture based on LSTM and CRF that obtained state of the art NER performance for English, German, Dutch and Spanish. Results are presented for CoNLL-2002 [Tjo02] and CoNLL-2003 [SD03] datasets. This approach uses both word-level embeddings as well as character-level embeddings. “The embedding for a word derived from its characters is the concatenation of its forward and backward representations from the bidirectional LSTM.” [LBS$^+$16, p.6]. For the model to consider both the word-level and character-level representations, dropout training was necessary and severely improved the performance. A different approach inspired on shift-reduce parsers is also presented, but performed worse than the LSTM-CRF architecture.

Gillick *et.al.* [GBVS15] use a different approach to text representation: text is represented as a sequence of bytes. This notation has a very small vocabulary (just the number of different characters) and results in a compact model when compared to models that use different word representation techniques such as feature sets or word embeddings, but still produces results similar to or better than state-of-the-art in POS tagging and NER [GBVS15]. The LSTM-based model does not depend on any of the standard tasks of the natural language processing pipeline, not even tokenization, which is required by most other models. The output of the model are annotations in the form [start, length, label] that in the case of NER correspond to the byte in which the named entity starts, the length of the named entity in bytes and the corresponding named entity class.

Chiu *et.al.* [CN15] make use of a bidirectional LSTM and CNN architecture to create a NER model that still is the state of the art model for the OntoNotes 5.0 dataset [HMP$^+$06]. The CNN architecture is used to obtain character level word representations that are concatenated with the word embedding representation of the word and input to the bidirectional LSTM. The best performing variation of the model includes using pre-trained embeddings obtained with the word2vec architecture and lexicon features.

Ma *et.al.* [MH16] use a similar approach to the work of Chiu *et.al.*. Just like Chiu *et.al.*, the main components of the model are a bidirectional LSTM and CNN architecture. This model is truly end-to-end, meaning it requires no feature engineering and proved to work with two separate sequence labelling tasks, obtaining state of the art performance for both POS and NER. Just like *Chiu et.al.*, Ma *et.al.* use the CNN to create character level representations of words and concatenate them with the word embedding representation before feeding the representations to the bidirectional LSTM. Dropout layers are included in multiple parts of the model to mitigate overfitting. Dropout is applied before character embeddings are input to the CNN and to the input and output vectors of the bidirectional LSTM.

All the tools presented for the Portuguese language (CitiusTools, FreeLing and spaCy) also support English, but there are some tools that do not offer support for Portuguese that are worth mentioning.

Stanford CoreNLP [MSB$^+$14] provides a set of human language technology tools, focused on ease of use. The CoreNLP tool is highly flexible and extensible. NLP tasks supported include POS tagger, NER, co-reference resolution, sentiment analysis, among others. The tool is optimized for the English language but also supports Arabic, Chinese, French, German and Spanish. Named entities are recognized using a combination of CRF sequence taggers trained on various corpora [FGM05]. Numerical entities are recognized using two rule-based systems, one for money and numbers and a separate system for processing temporal expressions [CM12].

Apache OpenNLP[6] is a machine learning based toolkit for the processing of natural language text. It supports the most common NLP tasks, such as tokenization, sentence segmentation, POS tagging, named entity extraction, chunking, parsing and co-reference resolution. For the task of NER, a maximum entropy model is trained. Trained models are available for English, Spanish and Dutch.

Natural Language Toolkit (NLTK) is an adequate tool both for teaching and working with computational linguistics. An extensive book written by the creators of NLTK is available [BKL09] in which they give a detailed description of NLTK's features. NLTK provides easy to use interfaces to over 50 corpora and lexical resources. NLTK provides a NER classifier trained for the English language. NLTK features available for the English language include: dataset loaders, multiple word classifier models, chunking models, NER models and more.

[6]http://opennlp.apache.org/

Chapter 3

Datasets

As mentioned before, it is hard to obtain annotated data for Portuguese and very few annotated datasets are freely available. Two different categories of datasets are explored in this work: annotated datasets and non annotated datasets. Annotated datasets are used to train and test the different deep learning models (described in Chapter 4) while non annotated data is used for the bootstrapping experiment. The bootstrapping technique (described in Chapter 5) can be used to include non annotated text in the training of NER models. This chapter includes details about both the annotated data and raw data used.

3.1 Data Format

Most of the time, available annotated datasets follow different formats to represent the text and the annotations. For example the datasets that resulted from HAREM use a XML format while the WikiNER dataset follows a simple text notation where each word is followed by the POS tag and NER tag separated by a vertical bar: "João|NOM|B-PER".

In order to create models that could be trained with all datasets it is necessary to adapt the datasets to a uniform format. The adopted uniform format is the CoNLL-2003 format, this format is adopted by almost all researches in their implementations with clear examples being the work of Lample *et.al.* [LBS+16], Chiu *et.al.* [CN15] and Ma *et.al.* [MH16].

In the CoNLL-2003 [SD03] data format the file contains one word per line and empty lines represent sentence boundaries. After each word there is tag that indicates if the word belongs to an entity. This tag encodes both entity boundaries and the entity type. This format does not support overlapping named entities. An example of an annotated phrase in the CoNLL-2003 format is presented in Table 3.1.

The original format presented in [SD03] includes chunk tags and part of speech tags along side with the named entity tag but for these experiments the chunk tag and part of speech tag can be ignored as they are not used in the training or testing of the models. All tested models follow the ideology of being an end-to-end models, making use of word embeddings to represent the

U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

Table 3.1: Original CoNLL-2003 format using IOB1 tag scheme. From [SD03].

text allows the models to be completely independent from the output of other NLP tasks, namely chunking or POS tagging.

The datasets used in this work are distributed in different formats and all were transformed into the CoNLL-2003 format before being used to train the models. To transform the datasets into the CoNLL-2003 format it is necessary to first tokenize the text into sentences and tokenize each sentence into words.

3.2 Tag Representations

A tag representation scheme needs to be able to identify entity boundaries and the entity category. There are multiple tag representation schemes, the differences are in the way that entity boundaries are identified. In all schemes words tagged with *O* are outside of named entities and words tagged with *I-XXXX* belong to an entity of category *XXXX*. The differences between tagging schemes are as follows [Hak13]:

- IOB1: Tag *B-XXXX* is the beginning of a named entity of category *XXXX* that immediately follows another named entity of category *XXXX*
- IOB2: Tag *B-XXXX* is used at the start of every named entity.
- IOE1: Tag *E-XXXX* is used to mark the last token of a named entity immediately preceding another named entity of category *XXXX*
- IOE2: Tag *E-XXXX* is used at the end of every named entity.
- BIOES or IOBES: Tag *S-XXXX* is used to represent a named entity with a single token. Named entities longer than one token always start with *B-XXXX* and end with *E-XXXX*

These differences are described in Table 3.2

There is some evidence that the IOBES tagging scheme improves model performance [RR09, DLCT15]. But others have not found any significant improvement over the IOB schemes [LBS+16].

Scheme	A	imprensa	foi	inventada	na	Alemanha	por	John	Gutenberg	.
IOB1	O	O	O	O	O	I-LOC	O	I-PER	I-PER	O
IOB2	O	O	O	O	O	B-LOC	O	B-PER	I-PER	O
IOE1	O	O	O	O	O	I-LOC	O	I-PER	I-PER	O
IOE2	O	O	O	O	O	E-LOC	O	I-PER	E-PER	O
IOBES	O	O	O	O	O	S-LOC	O	B-PER	E-PER	O

Table 3.2: Example of different tagging schemes applied to NER. Categories: Location (LOC) and Person (PER)

3.3 Annotated data

Annotated NER datasets are hard to come by for the Portuguese language. The datasets of reference for the Portuguese language are the ones produced for the HAREM contest. Another dataset that is useful for this work is WikiNER which focuses on Wikipedia data. Both the HAREM datasets and the WikiNER dataset have been briefly presented in Chapter 2.

These datasets have their own data format and needed to be transformed into the CoNLL-2003 data format. All 3 datasets from HAREM – first HAREM GC[1] ($HAREM_{first}$), miniHAREM GC[2] ($HAREM_{mini}$) and second HAREM GC [3] ($HAREM_{second}$) – were parsed from the XML format to the CoNLL-2003 format. The CoNLL-2003 format does not support multiple categories for the same named entity; in the cases where the same tokens were marked with multiple categories the first listed category was chosen. This preprocessing decisions may be a source of noise to the dataset, but details on how to correctly process the HAREM XML format into the CoNLL-2003 were not available. Santos *et.al.* [dSG15] also use HAREM datasets but do not specify the preprocessing steps done.

In addition to these original datasets, a selective scenario was created from $HAREM_{first}$ called $HAREM_{first\_selective}$ and from $HAREM_{mini}$ called $HAREMmini\_selective$ which only includes 4 named entity categories: organization (ORG), abstraction (MISC), location (LOC) and person (PER). The selective scenario is created to be able to assess the impact of the number of different named entity categories on the performance of the models. The 4 categories chosen to be part of the selective datasets were selected based their distribution in the original dataset (ORG – 18.0%; MISC – 8.5%; LOC – 25.0%; PER – 21.5%) and on the entity categories present on the CoNLL-2003 NER datasets [SD03].

Following the same approach as Santos *et.al.* [dSG15], the $HAREM_{first}$ dataset is split into train set and development (or validation) set and the $HAREM_{mini}$ is used as the test set for some of the experiments performed. The development set contains the last 5% of the $HAREM_{first}$ dataset. When referring to the train or development subsections of the datasets the additional subscript *dev* or *train* are used.

[1]https://www.linguateca.pt/aval_conjunta/HAREM/CDPrimeiroHAREMprimeiroevento.xml
[2]https://www.linguateca.pt/aval_conjunta/HAREM/CDPrimeiroHAREMMiniHAREM.xml
[3]https://www.linguateca.pt/aval_conjunta/HAREM/CDSegundoHAREM.xml

WikiNER is a *silver-standard* annotated dataset that resulted from the work of Nothman *et.al* [NRR+13]. Nothman *et.al* focused on automatically creating an annotated dataset by exploiting the text and structure of Wikipedia. The Portuguese Wikipedia dump used to create WikiNER was created on August 4, 2010. WikiNER is available in a custom format and needs to be transformed into the CoNLL-2003 format. The original WikiNER format follows the structure *word|POS tag|NER tag* and uses the IOB1 tag scheme. An example of the original format of WikiNER:

```
Astrobiologia|NOM|O é|V|O o|DET|O estudo|NOM|O do|PRP+DET|O
advento|NOM|O e|CONJ|O evolução|NOM|O os|DET|O sistemas|NOM|O
biológicos|ADJ|O no|PRP+DET|O universo|NOM|O
```

In the bootstrapping experiments (described in chapter 5, annotated datasets are used to evaluate the quality of the model trained at different bootstrapping iterations. Two annotated datasets are used for two different experiments: for the experiments involving news articles the $HAREM_{second}$ is used, while for the experiments involving Wikipedia text the WikiNER dataset is used. Since the bootstrapping experiments only focus on one named entity category, the category PERSON, it is necessary to remove all other entities from the annotated datasets used to test the model. The datasets that contain only named entities of the category PERSON are identified by the additional subscript *per*.

A very small annotated dataset, $News_{test\_per}$, was produced to evaluate the bootstrapping experiments. This small dataset is composed of 15 news articles from March of 2018 and only contains named entities of category PERSON. It was created with the intention of assessing the factor of dataset age from the performance measures of models trained with the bootstrapping method. This dataset was created specifically for the bootstrapping experiments because the textual data used to train the models is very recent, from 2017 and 2018, while the dataset used to test it, $HAREM_{second\_per}$, is from 1997.

Detailed statistics about all the annotated datasets used in this work are available in Table 3.3.

3.4 Raw data

The main goal of this work is testing supervised deep learning techniques to train NER models, to perform supervised learning annotated datasets are required but there are ways to explore non annotated data. All the deep learning models used have word embeddings as inputs and as explained and observed in chapter 4 initializing the DL models with pre-trained embeddings leads to major improvements in performance. To obtain pre-trained word embeddings large amounts of raw textual data are required, another useful application of raw data is training NER models using the bootstrapping process (details in chapter 5).

The raw data used in this work is divided into two textual genres: news articles and Wikipedia articles. The news articles are from various Portuguese newspapers and were fetched using the

Dataset	Number Tokens	Total number of entities	Number of documents	Number of entity categories
$HAREM_{first}$	92 228	4 972	129	10
$HAREM_{first\_train}$	87 594	4 805	–	10
$HAREM_{first\_dev}$	4 634	167	–	10
$HAREM_{first\_selective}$	92 228	3 578	129	4
$HAREM_{first\_selective\_train}$	87 594	3 458	–	4
$HAREM_{first\_selective\_dev}$	4 634	120	–	4
$HAREM_{mini}$	62 440	3 624	128	10
$HAREM_{mini\_selective}$	62 440	2 507	128	4
$HAREM_{second}$	84 300	7 030	129	10
$HAREM_{second\_per}$	84 300	1 965	129	1
$WikiNER$	3 499 683	263 732	892 834	4
$WikiNER_{per}$	3 499 683	64 078	892 834	1
$News_{test\_per}$	5 988	94	15	1

Table 3.3: Details of all annotated datasets used in this work.

SAPO Labs[4] platform, a portal that aggregates news from several news providers in Portugal. Wikipedia articles were obtained from the Portuguese Wikipedia dump of April 1, 2018.

The company responsible for hosting Wikipedia, Wikimedia, provides dumps of all Wikipedias at least once a month. The dump is a large file written in a wiki markup language, Wikitext[5]. This format includes all necessary information to format a Wikipedia page, figures, tables, titles, sections and so on. To extract the raw textual data it is necessary to parse the wiki markup language and remove all the meta-data and structuring elements.

Two different parsers were used to obtain the cleaned textual data from the dump file: WikiExtractor[6] and an adapted version of the Perl script written by Matt Mahoney[7]. WikiExtractor outputs multiple files, each with multiple articles containing only text without any of the Wikitext meta-data and structure, no tables, images, references to other articles, etc. The resulting files are then combined to obtain a single file of uninterrupted text. This dataset will be referenced as $Wiki_{raw}$. The perl script parser is used to obtain a text file to be used for training embeddings. Just like WikiExtractor, this parser excludes all Wikitext meta-data and structure but also transforms all words to lower case and all digits into their word representation. Despite excluding images and links, all captions are preserved. All words are lower cased in order to reduce the vocabulary size and maximize training instances for words when training embeddings. The resulting dataset is

[4]http://labs.sapo.pt/
[5]https://www.mediawiki.org/wiki/Wikitext
[6]https://github.com/attardi/wikiextractor
[7]http://mattmahoney.net/dc/textdata.html#appendixa

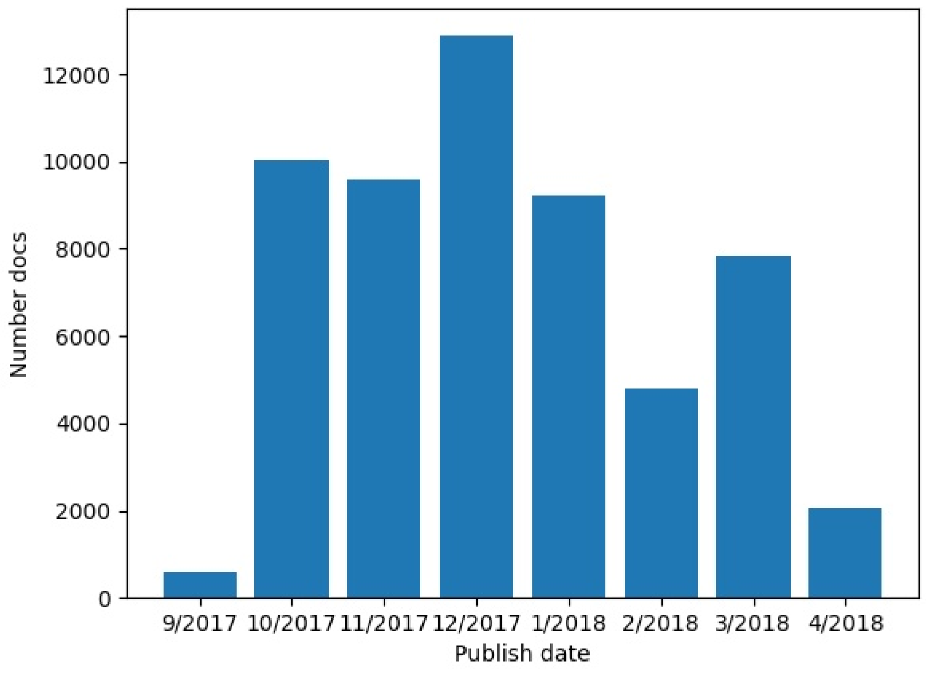

Figure 3.1: Publish date distribution of $News_{total}$ corpus.

referenced as $Wiki_{stripped}$.

For the bootstrapping experiments only a small portion of the $Wiki_{raw}$ dataset is used – the first 7 million tokens. It was necessary to restrict the dataset size because training times would be too large to obtain results in a feasible time, the 7 million number was chosen as it was the threshold where the dataset stopped fitting into GPU memory. This small portion is referenced as $Wiki_{partial}$.

The news dataset is composed of news articles from Portuguese news websites. All articles were published between September 2017 and April 2018 (see Figure 3.1). The articles were fetched using the Sapo Labs platform and come in the JSON format; a parsing script was developed to strip all the structural elements and meta-data, leaving only the title and body of the news article. The totality of the dataset contains 57 000 news articles but for the bootstrapping experiments only a subset of the first 15 100 news articles is used as the training set, the reason for this restriction is, just like in the $Wiki_{raw}$ dataset, GPU memory. The complete dataset is referenced as $News_{total}$ and the subset as $News_{partial}$

Statistics regarding the number of tokens and documents present in each raw text dataset is shown in Table 3.4.

Dataset	Number of Tokens	Number of documents
$Wiki_{raw}$	236 357 457	892 834
$Wiki_{stripped}$	422 023 462	892 834
$Wiki_{partial}$	7 000 000	-
$News_{total}$	23 307 376	57 000
$News_{partial}$	6 549 587	15 100

Table 3.4: Raw dataset details

Chapter 4

Deep Learning Models

This chapter focuses on presenting, in detail, all the architectures used to train embeddings and NER models for the Portuguese language. All models are heavily based on published work that focuses on the English language.

4.1 Embeddings

As mentioned in Chapter 2, embeddings are responsible for significant improvements in many NLP tasks. Embeddings provide an effective process to create vectorial representations of words. These representations have two major advantages over one-hot encodings, most notably: the vectorial space dimensionality is limited and not dependant on the size of the vocabulary; some of the relations between words are preserved in vectorial form.

Embeddings are most commonly used to represent words, but some research has been made with character embeddings. Examples of architectures that make use of character embeddings include the work of Santos *et.al.* [SZ14, dSG15], the work of Chiu *et.al.* [CN15], and the work of Xuezhe Ma *et.al.* [MH16].

The quality of the embeddings is related with the amount of data used to train. Consequently, training high quality embeddings for languages that have textual resources readily available is much easier. There are research groups and companies that provide pre-trained embeddings; sadly, most of these embeddings are for the English language.

For this work, word embeddings were trained from scratch using Portuguese Wikipedia data. Two different embedding training architectures were used: *word2vec* [MCCD13] and *wang2vec* [LDBT15], a slight modification of *word2vec*. Details about these and other embedding models are highlighted in Chapter 2.

The work of Hartmann *et.al* [HFS+17], which focuses on evaluating different embedding models for the Portuguese language, highlighted the *wang2vec* structured *skipngram* embedding model to be the best performing in extrinsic evaluation for the tasks of part of speech tagging and semantic similarity. Extrinsic evaluation consists on using the embeddings that are meant to be evaluated

Embedding Model	Vocabulary Size	Dimension	Variation	Window Size
$wang2vec_{64D}$	615 108	64	structured skipgram	8
$wang2vec_{100D}$	615 108	100	structured skipgram	8
$word2vec_{100D}$	610 395	100	cbow	8

Table 4.1: Trained word embeddings details.

on a specific task, for example POS or NER, and analyse the impact on the overall task performance. Good quality embeddings will improve overall model performance while lower quality embeddings will hinder the model's performance.

To train embeddings using the *word2vec* model, the original implementation written in C[1,2] is used. To train embeddings using the *wang2vec* model, an implementation written in C[3] is used. The reason behind choosing these models to train embeddings is the work of: Santos *et.al.* [dSG15]; Chiu *et.al.* [CN15]; Lample *et.al.* [LBS+16]; Ma *et.al.* [MH16]. These researchers obtained the best results in their NER models using one of the embeddings architectures mentioned therefore using them seems like an obvious choice.

For all embedding models the training dataset was the $Wiki_{stripped}$ raw dataset (see Chapter 3). Resulting embeddings were informally manually evaluated using intrinsic evaluation through word analogies[4]. This intrinsic evaluation was merely a sanity check to confirm that the learned embeddings captured relationships between words, the true test to their value is by doing extrinsic evaluation which is using them in training NER models. Details of the trained embeddings can be seen in Table 4.1, embedding dimensions were chosen based on the work of Lample *et.al.* [LBS+16] where embeddings of dimension 100 are used for the English language and embeddings of dimension 64 are used for Spanish, Dutch and German.

4.2 NER Architectures

In this section all NER models used in this work are described in detail. One of the presented models has already been applied to the Portuguese language before in the work of Santos *et.al* [dSG15], but the other three were applied to the Portuguese language for the first time during this work.

Deep learning architectures are sophisticated neural networks with many layers and many units per layer so that they are able to model complex functions that perform different tasks such as image classifications or speech recognition. For NLP, namely NER, one of the most used architectures is the recurrent neural network and it's variations. RNNs can effortlessly handle variable length sequences and create sequence to sequence models. A sequence to sequence model is one of the possible ways to model the NER problem, a sequence of words is input to the model and a sequence of the respective NER tags is output from the model.

[1]https://code.google.com/archive/p/word2vec/
[2]https://github.com/dav/word2vec
[3]https://github.com/wlin12/wang2vec
[4]https://github.com/nlx-group/lx-dsemvectors/blob/master/testsets/LX-4WAnalogies.txt

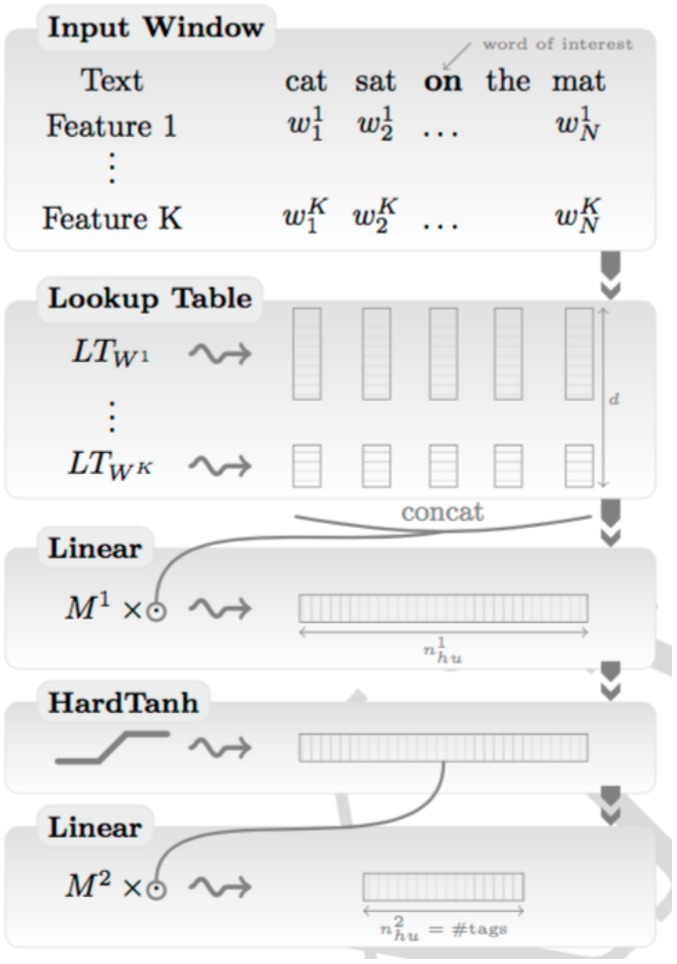

Figure 4.1: Window approach network structure. From [CWB+11].

All networks used in this work are a replica or a copy with very few modifications of architectures that have proven to be successful at performing NER in the English language. In order to accelerate the experiments most of the hyper-parameters of the network are set to those reported to be the best for English datasets; this may limit the performance on Portuguese data. Tuning all hyper-parameters to the best combination for the Portuguese datasets is possible but requires running a high number of experiments and due to time limitations that would restrict the amount of architectures that could be explored.

In the following subsections each of the architectures explored in this work is described in detail. The names given to the architectures were chosen based on their characteristics, these names are not the ones original given by the authors when introducing the architectures.

4.2.1 $Window_{Network}$

Introduced by Collobert *et.al.* [CWB+11], the $Window_{Network}$ was the architecture that shifted the focus of NLP tasks to Neural Networks and Deep Learning. The $Window_{Network}$ focuses on the *window* approach network that is summarized in Figure 4.1.

The work of Collobert *et.al.* was the first time the choice of features was a completely empirical process and not a list of hand-picked features that relied on linguistic intuition and mostly trial and error. Hand-picked features have the added restriction that they are highly task dependant meaning for every task a new set of features would need to be developed. Collobert *et.al.* incorporate the feature engineering process into the network itself. The architecture takes the input sentence and learns several layers of feature extraction to process the inputs.

The way this architecture manages to represent word context is by using windows: when trying to predict the tag for a specific word the input contains not only the target word but also its neighbours. The window size is one of the hyper-parameters of the network.

The first two layers are responsible for automating the “feature engineering process”: the first layer extracts features from single words while the second layer extracts features from the window of words. This means that this network architecture is not tied to any specific NLP task but can adapt to multiple tasks seamlessly – the features that matter for one tasks might not matter for another but the network adapts automatically.

The first layer, identified as Lookup Table in Figure 4.1, is standard for all other architectures explored in this work; it is the layer that maps words into their vectorial representation. When using pre-trained embeddings, the only difference is that this first layer is initialized with the pre-trained weights instead of random initialization.

The remaining layers of this architecture are quite simple: two linear layers with an HardTanh activation layer in between. A linear layer can be summarized in the following way: given a fixed size vector r the output vector o of the linear layer will be $o = Wr + b$ where W represents the layer weights and b the layer bias; W and b are the parameters to be learned for the linear layer. The HardTanh activation layer is applied element wise to each input vector dimension:

$$x = \begin{cases} -1 \quad if x < -1 \\ x \quad if -1 <= x <= 1 \\ 1 \quad if x > 1 \end{cases}$$

The last layer maps the internal representation of the network to the tag space: the output has a dimension equal to the number of possible different tags. Each dimension of the output vector can be interpreted as the *score* of the corresponding tag.

Two different score functions are presented by Collobert *et.al.*: *word-level log-likelihood* and *sentence-level log-likelihood*. The log-add operation is defined as

$$\underset{i}{logadd} z_i = log(\sum_i e^{z_i}) \tag{4.1}$$

In the word-level log-likelihood score function each word is considered independently and for one training example (x,y) can be calculated as follows:

$$logp(y|x,\theta) = [f_\theta]_y - \underset{j}{logadd}[f_\theta]_j) \tag{4.2}$$

where $[f_\theta]_i$ is the score of the i^{th} tag given input x and network parameters θ.

This score function, commonly known as *cross-entropy*, may not be appropriate for the NER problem. Assuming that the tag of a single word is independent from the tags of the words surrounding it may lead to weaker results. The tag schemes described in Chapter 3 do not allow certain sequences of tags to appear. For example, considering the tag scheme IOBES we can never have a *B-XXX* tag following a *I-YYY* tag.

To include the dependencies between word tags in a sentence, the sentence-level log-likelihood score function is used. This score function requires some additional network parameters: a transition score $[A]_{i,j}$ for jumping from i to j tags in successive words; initial score $[A]_{i,0}$ for starting from the i^{th} tag. The score of a sentence $[x]_1^T$ along the path of tags $[i]_1^T$ is given by the sum of transition scores and network scores:

$$s([x]_1^T,[i]_1^T,\theta) = \sum_{t=1}^{T}([A]_{[i]_{t-1},[i]_t} + [f_\theta]_{[i]_t,t}) \tag{4.3}$$

where $[f_\theta]_{[i]_t,t}$ represents the score output by the network with parameters θ, for the sentence $[x]_1^T$ and for the i^{th} tag at the t^{th} word. The sentence level cost function can then be given by:

$$logp([y]_1^T|[x]_1^T,\theta) = s([x]_1^T,[y]_1^T,\theta) - \underset{\forall [j]_1^T}{logadd}\, s([x]_1^T,[j]_1^T,\theta) \tag{4.4}$$

The number of operations to calculate this sentence-level cost function grows exponentially with sentence length, as all the combination of tag sequences need to tested. However it is possible to compute it in linear time [CWB+11].

During training, the objective is to maximize the log-likelihood (Eq. 4.4) over all the training pairs. When testing, given a sentence $[x]_1^T$ to process, it is necessary to find the path that minimizes the sentence score. The Viterbi algorithm [For73] is used.

The $Window_{Network}$ was also used and modified by Santos *et.al.* [dSG15]. The modifications were mainly centered in including character level representations of words. Character representations are obtained using a small network composed of a character embeddings layer, a linear layer and a max-polling layer.

Collobert *et.al.* also highlight that despite the network being able to perform the feature engineering automatically it can be useful to include some manually selected features. The manually selected features include the capitalization feature and the suffix feature. The capitalization feature has five possible values: all letters lower-cased; first letter upper-cased; all letters upper-cased; word contains an upper-cased letter; all other cases. The suffix feature has one hyper-parameter, the suffix size. A suffix size of 3 means that the last three characters of the word are used as a feature. Both the capitalization and suffix embeddings have dimension five, this means each of the

Parameter Name	Value
Hidden size	300
Window size	5
Extra feature emb. size	5
Suffix size	3
Learning rate	0.01
Batch size	16

Table 4.2: Hyper parameters of $Window_{Network}$.

features is represented by vector of dimension five. Two variations of the $Window_{Network}$ are used to run experiments with, one excluding the extra features and another including both the capitalization and suffix features. The network without the extra features is named as $RawWindow_{Network}$ and the one that includes the capitalization and suffix features is named $CompleteWindow_{Network}$

The various hyper-parameter values for the $Window_{Network}$ are listed in Table 4.2.

4.2.2 $BiLSTM_{Network}$

The second network architecture used to run experiments on Portuguese NER is based on the work of Chiu *et.al.* [CN15]. Chiu *et.al.* use a bidirectional LSTM-CNN architecture to perform NER that can be summarized by Figure 4.2. The $BiLSTM_{Network}$ used in the experiments follows the same structure but discards the CNN-extracted character features, so as to be able to compare the $Window_{Network}$ with a bidirectional LSTM network. Removing the character level word representations produced with the CNN in the recurrent neural network, $BiLSTM_{Network}$, is done to provide a fair comparison with feed forward neural network, $Window_{Network}$, that has no character level word representation.

The first layer is responsible for transforming words into their vectorial representation just like in the $Window_{Network}$. After having the word representation the word is fed both to a forward and a backward LSTM. At each time step, the outputs of both the forward LSTM and backward LSTM are fed into a linear layer followed by a LogSoftmax layer and finally added together to generate the final tag scores (see Figure 4.3). The network works on a sentence level, which means that the cell state stores information about the whole sentence; thus, word tags are predicted based on the whole sentence and not only on a restrict number of neighbouring words like in the window approach network presented above.

The idea behind using two LSTM networks, a forward LSTM network that processes the sentence from start to finish and a backwards LSTM network that processes the sentence from finish to start is to ensure that the prediction of the tag for a specific word is based on information about the words that precede and that follow it. This setup means that the tag for the n^{th} word in a sentence is predicted based on information of the words preceding it, from word 0 to word $n-1$,

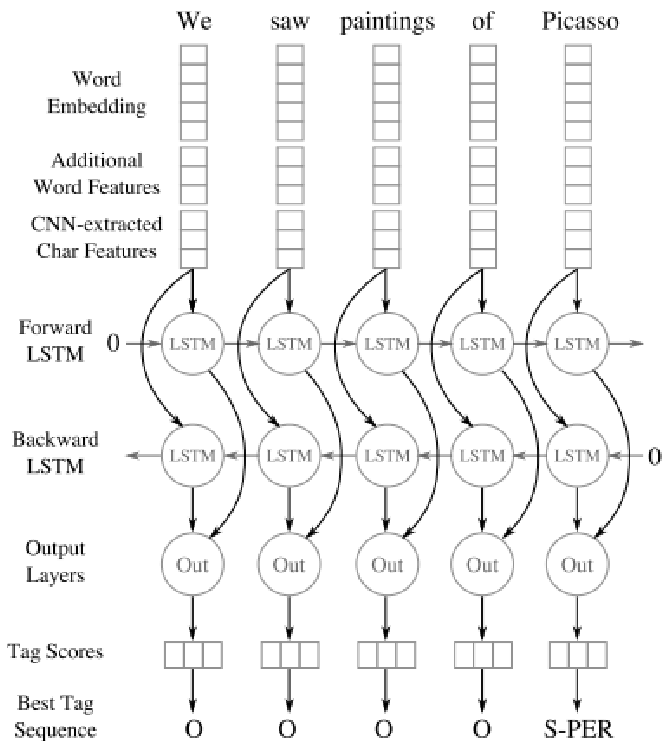

Figure 4.2: Bidirectional LSTM-CNN network structure. From [CN15].

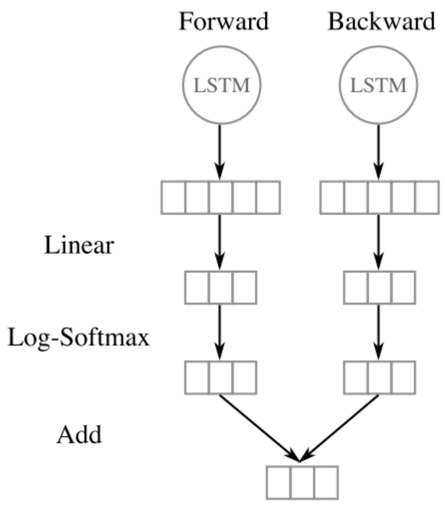

Figure 4.3: Last layer of Bidirectional LSTM-CNN network. From [CN15].

Parameter Name	Value
Batch size	16
LSTM state size	275
LSTM layers	1
Dropout	0.68
Learning rate	0.01

Table 4.3: Hyper parameters of $BiLSTM_{Network}$.

encoded by the forward LSTM network, but also based on the words following it, from word $n+1$ to the last word in the sentence, encoded by the backward LSTM network.

In order to reduce overfitting, dropout is applied after the output of each LSTM layer; dropout is only applied during training. During test time the dropout is not performed, meaning all the outputs of both LSTM are preserved.

The objective function used in this architecture is the same as the one used in the $Window_{Network}$, meaning the transition score matrix and initial score matrix are added to the network parameters.

The various hyper-parameter values for the $BiLSTM_{Network}$ are listed in Table 4.3.

4.2.3 $BiLSTMChar_{Network}$

The $BiLSTMChar_{Network}$ was introduced in the work of Lample *et.al.* [LBS+16] and just like the $BiLSTM_{Network}$ it uses a one layer bidirectional LSTM network to create a sequence to sequence model. A sequence of words goes in and a sequence of NER tags comes out.

The outputs of both LSTM networks are combined by concatenating the outputs and feeding them to a linear layer that outputs a vector with half the size; the activation function used is the hyperbolic tangent. The output of the linear layer is finally fed to the final layer, a linear layer without an activation function, that maps the internal representation to a vector with the number of dimensions equal to the number of different tags in the dataset. Lample *et.al.* call this the CRF layer.

Just like all the other network architectures no language-specific or task-specific resources are used, but this architecture goes a step beyond: no hand-picked features are used. In order to improve the word representation, character-based models of words are created and used side by side with word embeddings.

Character-based models of words are created to be able to represent a single word based on the characters that compose it. This representation is combined with the word embedding representation to generate the final word representation that is used as input for the sequence labelling network. A diagram of the process of generating the final word representation can be seen in Figure 4.4. The character representation of a word in the $BiLSTMChar_{Network}$ is created by feeding a word char-by-char to a Bidirectional LSTM and at the end concatenating the outputs of both the forward and backward LSTM.

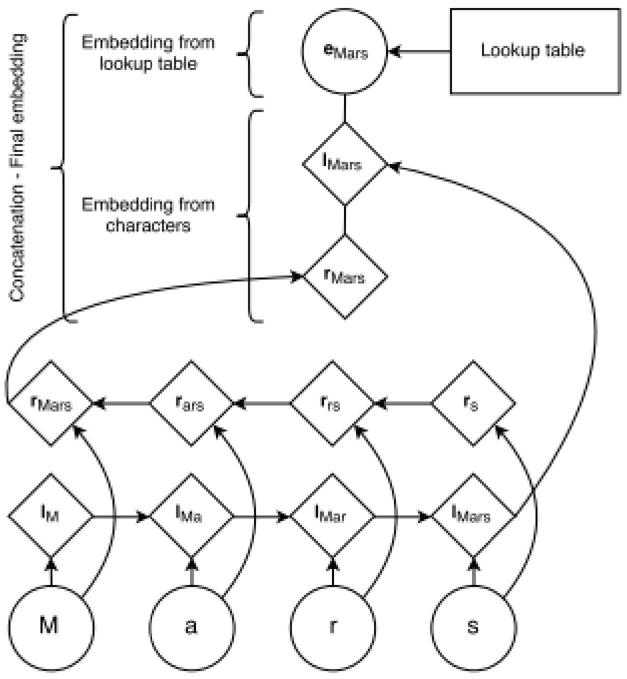

Figure 4.4: Full representation of the word "Mars", including character representations and word embeddings. The l_{Mars} label represents the output of the forward character LSTM and the r_{Mars} label represents the output of the backward character LSTM. From [LBS+16].

Just like the $BiLSTM_{Network}$, this architecture also uses dropout but for a different reason and in a different place. Dropout is applied after the final embeddings layer just before the input to the bidirectional LSTM. According to Lample *et.al.* [LBS+16], this was necessary because the incorporation of character-level embeddings were not improving the overall performance; applying dropout encourages the model do depend on both the word embeddings and character-level representations and led to significant performance improvements.

Lample *et.al.* call the last layer of the network a CRF layer, this last layer is a linear layer that maps the internal representation to a vector with dimension equal to the number of different tags. This vector represents the score of each tag and is used to calculate the score function. The score function is the same as the two previous architectures, where the objective is to maximize the sentence-level log-likelihood.

Just like in the other architectures it is necessary to add some parameters to the network, the transition score matrix and initial score list. Lample *et.al.* decide to combine the initial score list into the transition matrix, *start* and *end* tags were also added meaning that the transition matrix **A** is a square matrix of size $k+2$ where k represents the number of different NER tags in the dataset.

During training, the objective is to maximize the log-probability of the correct tag sequence, given by Equation 4.4, and when decoding the objective is to find the path that minimizes the sentence score.

Parameter Name	Value
LSTM state size	100
Batch size	16
Char Embedding dimension	25
Char LSTM hidden size	25
Dropout	0.5
Learning rate	0.05

Table 4.4: Hyper parameters of $BiLSTMChar_{Network}$.

The code used to run the $BiLSTMChar_{Network}$ experiments was not written from scratch, the implementation made available by Lample *et.al.*[5] is used.

The various hyper-parameter values for the $BiLSTMChar_{Network}$ are listed in Table 4.4.

4.2.4 $BiLSTM\_CNN_{Network}$

The $BiLSTM\_CNN_{Network}$ was introduced in the work of Ma *et.al.* [MH16]. This architecture includes a bidirectional LSTM to create a sequence to sequence model, a CNN to extract character-level features from words and a CRF layer to jointly decode the best chain of NER tags for a given sentence.

The overall architecture structure is quite similar to the $BiLSTMChar_{Network}$: they both use bidirectional LSTMs and include a character-level word representation in the input vector for each word. Major differences between these two architectures include: the way character-level representations of words are created; where to apply dropout.

To create character representations, Ma *et.al.* make use of a simple CNN whose structure can be seen in Figure 4.5. Similar approaches to creating character-level word representations had already been developed by Chiu *et.al.* [CN15]. First the word is padded depending on the CNN window size, then each character of the word is transformed into its embedding representation. Before applying the convolution, dropout is applied to the character embeddings.

A convolution is applied to the character embedding matrix, where the convolution kernel is equal to *[CNN_WINDOW_SIZE, CHAR_EMBEDDING_DIMENSION]*. After the convolution, a matrix with dimensions *[WORD_LENGTH, NUMBER_FILTERS]* is obtained. To transform this matrix into the character level word representation, a max-polling is used to transform the matrix of size *[WORD_LENGTH, NUMBER_FILTERS]* into a vector of size *[NUMBER_FILTERS]*.

Dropout is applied to the final word representation before the bidirectional LSTM and to the output of the two LSTMs. Dropout is applied to avoid overfitting the data. Ma *et.al.* report major improvements when using the described dropout setup in both NER and POS tasks.

The learning method is stochastic gradient descent just like all the other models but with two slight modifications: gradient clipping and variable learning rate. The original learning rate, η_0 is

[5]https://github.com/glample/tagger

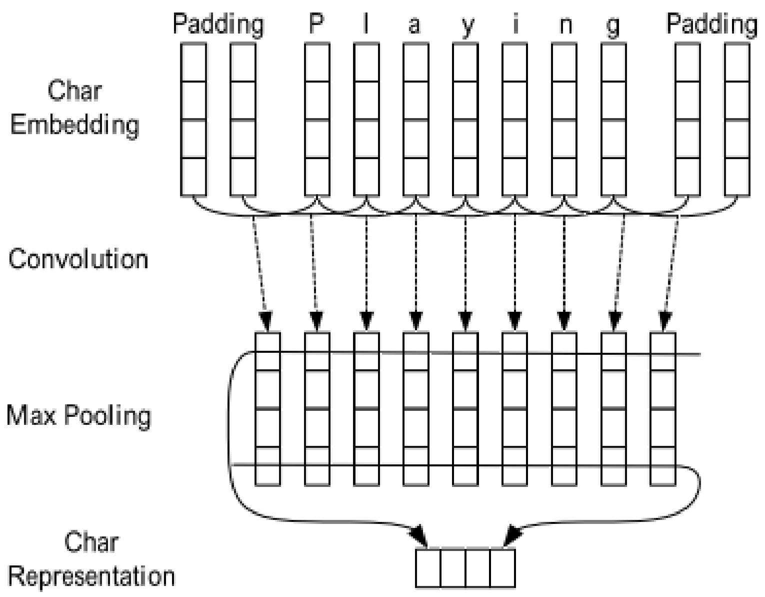

Figure 4.5: CNN used to extract character level representation of the word *Playing*. Dashed lines represent that dropout is applied. From [MH16].

updated using the rule $\eta_t = \eta_0/(1+\rho t)$, where ρ is the decay rate and t represents the number of epochs completed. All gradients are clipped to the value of 5.0 to reduce the effects of the gradient explosion problem.

The score function used for training the model is the exact same as in the $BiLSTMChar_{Network}$, maximizing the sentence-level log-likelihood.

Ma *et.al.* made available their implementation[6], written in Python 2.7 with the Pytorch deep learning framework. All experiments involving the $BiLSTM\_CNN_{Network}$ are ran using this implementation.

The various hyper-parameter values for the $BiLSTM\_CNN_{Network}$ are listed in Table 4.5.

4.3 Challenges

In order to get a righteous sense of the viability of using deep learning models for NER in the Portuguese language this work explores multiple NER deep learning architectures. As a consequence, all the work related with tuning a network is multiplied by the number of networks explored. When training a network, depending on the architecture, there are many hyper parameters that need to be tuned for the network to achieve top performance. From all the architectures explored, only the $Window_{Network}$ was tuned for Portuguese textual data [dSG15]; all others were tested and tuned on the English language.

[6]https://github.com/XuezheMax/NeuroNLP2

Parameter Name	Value
Batch size	16
Decay rate	0.05
Initial learning rate	0.01
LSTM hidden size	256
LSTM layers	1
Char embedding dimension	30
Number of CNN filters	30
CNN window size	3
CNN stride	1
Dropout	0.5

Table 4.5: Hyper parameters of $BiLSTM\_CNN_{Network}$.

There are different strategies to identify better hyper parameters for the networks but all require running a full training cycle of the network. This limits the number of hyper parameter combinations that can be tested, since training a network from scratch takes a significant time and can vary greatly depending on architecture, hyper-parameters and data used.

As mentioned before most hyper parameters of the architectures used were set to the hyper parameters reported in the paper for that specific architecture. The reported hyper parameters are tuned for the dataset where they have been tested, normally datasets in the English language, and there is no guarantee of optimality when used with different datasets or different languages. Tests were done where some of the hyper-parameters were arbitrarily changed but all the changes done affected the performance of the resulting model negatively.

4.4 Implementation

The python ecosystem for data science is very extensive and many libraries and frameworks are available. Choosing the tools to work with is, most of the times, a matter of preference. This chapter lists and describes the libraries and frameworks chosen to set up the experiments and pre-process the data. All code developed for this work is publicly available in Github[7].

4.4.1 Pytorch

The deep learning framework used to implement the NER models is PyTorch[8]. PyTorch is a python package that supports tensor computation with GPU acceleration and includes automatic differentiation tools required for deep learning model training. PyTorch stood out after testing multiple DL frameworks due to it's simplicity and very granular controls that allow for specific

[7]https://github.com/ivoadf/PT_NER_DL

[8]https://pytorch.org/

tuning and inspecting of models. The fact that it allows the user to implement networks at a very low level removes the black box feeling that some DL frameworks have and helps better understand the whole process.

4.4.1.1 Description

Unlike Tensorflow[9], Theano[10] or Caffe[11] which have a static view of the world, PyTorch uses a dynamic representation of networks. Dynamic representations allow the user to modify the network without having to start from scratch. Dynamic representations allow for easier debugging of network training, allowing the user to inspect the values and gradients of each specific tensor.

PyTorch supports execution of models both on CPU and GPU, including the execution of models distributed between multiple GPUs.

4.4.1.2 Autograd

One of the most important features of a deep learning framework is automatic differentiation. This means that the user can create networks using PyTorch tensors and operations, no matter how complex, and have the gradients for each network parameter calculated automatically. In PyTorch the library responsible for automatic differentiation is *autograd*.

4.4.1.3 GPU integration

As mentioned before, moving a model from executing on CPU to executing on the GPU is seamless, but not all models are worth training on GPUs. My experiments show that a model with around 1 million trainable parameters is not worth training on GPU while a model with around 60 million trainable parameters sees a significant performance improvement when ran on a GPU.

The reason not all models run faster on the GPU and sometimes run even slower is due to the fact that transferring training data to the GPU and then transferring results back to RAM has a significant overhead. With a small model the speed up obtained by executing on the GPU is not sufficient to overcome the data transfer overhead, but with larger models the GPU can execute the training step much faster than the CPU, overcoming the data transfer overhead and resulting in a smaller total training time.

This can be observed by running two NER models, one with a large number of trainable parameters and another with much less, results in Table 4.6. The larger model runs much faster on GPU and the smaller model runs faster on CPU. There are techniques to reduce or eliminate the data transfer overhead. In cases where the whole dataset fits into GPU memory it can be preemptively loaded, so that the GPU can process batches without having to wait after every batch for the next one to load into GPU memory.

[9]https://www.tensorflow.org/
[10]http://deeplearning.net/software/theano/
[11]http://caffe.berkeleyvision.org/

Iterations	Number parameters	Train time (s)	
		CPU	GPU
100 000	395 739	392	488
2 000	45 758 069	558	6.7

Table 4.6: CPU versus GPU training time comparison. One epoch with dataset of 100000 words.

4.4.2 Natural Language Tool Kit

The natural language tool kit (NLTK)[12] is an extensive library with built-in interfaces for multiple corpora and pre-trained models for many NLP tasks. NLTK is free and open-source, meaning it is used by people with all backgrounds: teachers, students, researchers and even in the industry.

4.4.2.1 Tokenizer

From all the endless features of NLTK, the one feature used for this project was the tokenizer. NLTK offers both a pre-trained word and sentence tokenizer. Both the sentence and word tokenizers were useful for preprocessing Portuguese datasets. As mentioned before, in chapter 3, to process textual data into the CoNLL-2003 format it is necessary to first tokenize the text into sentences and then into words.

4.5 Training

For all networks the learning algorithm used was mini-batch stochastic gradient descent. For some of the architectures we used variable learning rate and for others a fixed learning rate. For all experiments the mini-batch size used was 16, meaning that each update to the weights of the network is based on the combined loss function of 16 training examples.

4.6 Results

For all the experimental results described below, the models were trained and tested using two combinations of datasets described in Table 4.7. The selective dataset combination was created to assess the impact that the number of different named entity categories present in the dataset has on the performance of NER models. All metrics are obtained using the official CoNLL-2003 [SD03] evaluation script.

Analysing the obtained F1 measures (Tables 4.8 and 4.9) for the various experiments, some interesting observations can be extracted. Firstly, a more substantial difference between the top performing models on the *Complete* dataset combination and the *Selective* dataset combination was expected. The *Complete* dataset combination has a total of 10 different named entity categories, while the *Selective* dataset combination has only 4 different named entity categories. This

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

Combination Name	Train dataset	Dev dataset	Test dataset
$Complete$	$HAREM_{first\_train}$	$HAREM_{first\_dev}$	$HAREM_{mini}$
$Selective$	$HAREM_{first\_selective\_train}$	$HAREM_{first\_selective\_dev}$	$HAREM_{mini\_selective}$

Table 4.7: Dataset combinations used for experiments.

Model	Embeddings	$Complete$			
		F1	Precision	Recall	Epoch
$RawWindow_{Network}$	$Wang2Vec_{64D}$	35.37	46.42	28.57	16
$CompleteWindow_{Network}$	$Wang2Vec_{64D}$	43.26	42.40	44.15	14
$BiLSTM_{Network}$	$Wang2Vec_{64D}$	59.61	64.86	55.16	14
	$word2vec_{100D}$	55.05	62.16	49.40	10
$BiLSTMChar_{Network}$	—	54.86	57.70	52.29	89
	$word2vec_{100D}$	67.02	68.31	65.78	53
	$Wang2Vec_{64D}$	67.35	68.94	65.84	92
$BiLSTM\_CNN_{Network}$	$word2vec_{100D}$	68.52	71.57	65.71	59
	$Wang2Vec_{64D}$	**69.97**	**72.64**	**67.50**	87
	$Wang2Vec_{100D}$	53.72	64.88	45.83	61

Table 4.8: Experiment results with the complete set of datasets.

Model	Embeddings	$Selective$			
		F1	Precision	Recall	Epoch
$RawWindow_{Network}$	$Wang2Vec_{64D}$	34.62	45.64	27.88	3
$CompleteWindow_{Network}$	$Wang2Vec_{64D}$	47.13	52.81	42.55	26
$BiLSTM_{Network}$	$Wang2Vec_{64D}$	61.64	65.00	43.70	11
	$word2vec_{100D}$	46.54	57.22	39.22	28
$BiLSTMChar_{Network}$	—	52.08	54.31	50.02	13
	$word2vec_{100D}$	**70.15**	**71.90**	68.49	47
	$Wang2Vec_{64D}$	69.50	70.12	**68.89**	88
$BiLSTM\_CNN_{Network}$	$word2vec_{100D}$	49.04	54.18	44.79	198
	$Wang2Vec_{64D}$	68.44	70.67	66.35	205
	$Wang2Vec_{100D}$	11.70	19.36	8.38	241

Table 4.9: Experiment results with the selective set of datasets.

Model	This work			Santos *et.al.*		
	F1	Precision	Recall	F1	Precision	Recall
$RawWindow_{Network}$	35.37	46.42	28.57	57.84	63.32	53.23
$CompleteWindow_{Network}$	43.26	42.40	44.15	65.73	68.52	63.16

Table 4.10: Results observed in this work compared to those reported by Santos *et.al.* [dSG15], obtained with the *Complete* dataset combination.

intuitively suggests that models for the *Selective* dataset combination would have substantially higher performance measures. The measured difference does follow this intuition, but in a much smaller scale: the real difference between the best performing models for the two dataset combinations is under 1.0% of F1.

Just like previous published work [CWB+11, CN15, LBS+16, HXY15, LZWZ11, dSG15, SM13, YZD17, DMR+15, NRR+13, MH16], major improvements were observed when using pre-trained embedding weights to initialize the models. This was verified for both dataset combinations: improvements of up to 18.07% in the F1 measure were observed for the $BiLSTMChar_{Network}$.

The two variations of the first network, $RawWindow_{Network}$ and $CompleteWindow_{Network}$, were the only architectures that had a feed forward neural network architecture. All others include a sequence to sequence model based on recurrent neural networks. Looking at the results we can observe that the networks that use recurrent neural network architectures perform significantly better than the feed forward neural networks. This observation matches the latests developments in NLP where state of the art systems for many tasks were improved after using deep learning architectures to train new models. Deep learning architectures, namely recurrent neural networks, fit perfectly into the textual data domain and have proved to better capture and model the context of words, meaning better and more accurate predictions can be made.

In the work of Santos *et.al.* [dSG15], some of the experiments were ran using the same configuration as the $RawWindow_{Network}$ and the $CompleteWindow_{Network}$, with the same datasets. However, the results reported by Santos *et.al.* were better than those observed in our experiments. The differences in performance between the work of Santos *et.al.* and this work can be seen in Table 4.10. These differences may be the result of different preprocessing of the HAREM datasets or the use of better pre-trained word embeddings.

All the model architectures described and tested performed much better in the English language, in some cases showing differences in F1 score of up to 23%. The $BiLSTMChar_{Network}$ and $BiLSTM\_CNN_{Network}$ both have F1 scores above 90.0% [LBS+16, MH16] for the NER task in the CoNLL-2003 English dataset while in the *Complete* scenario neither one gets scores above 70% F1 (see Table 4.8). These abrupt differences in performance between the two languages can be due to a multitude of factors: the inherent differences between the two languages; the quality of the pre-trained word embeddings used to initialize the models; the hyper-parameters used were tuned for the English dataset; characteristics of available training datasets.

To better understand the difference in performance between the model trained using English

Variable	CoNLL-2003	HAREM I GC
Number tokens	204 567	92 228
Number sentences	14 987	3 682
Average number tokens per sentence	13.65	25.05
Number of entity categories	4	10

Table 4.11: Dataset statistics comparison between Portuguese HAREM I GC NER dataset and CoNLL-2003 English NER train dataset.

datasets and the same model trained using Portuguese datasets, a comparison between the datasets was done. In Table 4.11 it is possible to observe some clear differences between the datasets, namely, total size in terms of tokens and the average sentence length.

Average sentence length is an important characteristic because all the NER models used for this work are trained on a sentence level, that is, each training example is a complete sentence. Longer sentences mean that more tokens are processed before making a tag prediction. This increase in the number of tokens consumed by the model before prediction might be one of the reasons behind the drop in performance.

If we strictly look at the number of tokens in each dataset, the HAREM I GC seems to be almost half the size of the CoNLL-2003 English dataset. However, the models explored are trained on a sentence level, which means that the true number of training examples is the number of sentences in the dataset. In terms of number of sentences the HAREM I GC has only approximately 25% of the number of sentences present in the CoNLL-2003 English dataset, which is a big drop from when the datasets are compared using the number of tokens. This change in relative size is due to the fact that the HAREM I GC has an average number of tokens per sentence much higher than the CoNLL-2003 English dataset (as shown in Table 4.11.

Another important difference that most certainly has impact in model performance is the quality of the pre-trained word embeddings used. The English language has more raw textual data resources freely available than the Portuguese language. Large amounts of raw textual data are essential to obtain good quality embeddings. All the pre-trained word embeddings used in this work were trained using textual data from the Portuguese Wikipedia, a corpus with a total of around 422 million tokens, while pre-trained word embeddings for the English language are trained in billions of tokens. The publicly available GloVe[13] word embeddings for the English language were trained in a total of 6 billion tokens.

Lastly, another plausible reason behind the drop in performance is the fact that most of the hyper-parameters of the architectures tested were left unchanged from the ones reported in the original papers. This means that hyper-parameters that were tuned for English datasets were used in Portuguese datasets. Reasons behind choosing not to do hyper-parameter tuning specifically for the Portuguese dataset are discussed in Section 4.2.

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

Chapter 5

Bootstrapping

Bootstrapping can be a good way to take advantage of the large amounts of non annotated data available. The objective is to make use of this data to help train models without the need of annotated data. This chapter describes the process of bootstrapping that was followed in detail, including what datasets were used, their characteristics, the performance results of the bootstrapping experiments and what were the challenges found during the process. As explained in [TSO11] and illustrated in figure 5.1, bootstrapping consists on:

1. identifying named entities in an unannotated corpus using a dictionary-based approach
2. training a model with the newly annotated corpus
3. testing the trained model on an external annotated corpus
4. re-annotating the corpus using the model
5. repeat until performance measures drop

The steps described above are repeated until the stop condition is met. The stopping condition is the moment when the performance measures drop, any further iterations do not improve the model and the new names detected become false positives leading to a model that evolves to be worse and worse over time.

The fact that unannotated corpora can be included in training the model is a great advantage, as unannotated corpora makes up for the vast majority of all available corpora and is for the most part completely free.

Usually, named entity recognition focuses on multiple entity classes. However, to obtain the initial list of annotations necessary for bootstrapping it is necessary to identify a pattern that is followed by entities of that specific type. Identifying patterns that match enough entities with 100% precision is not always trivial. For these bootstrapping experiments, just like in [TSO11], only entities of the type *PERSON* are considered.

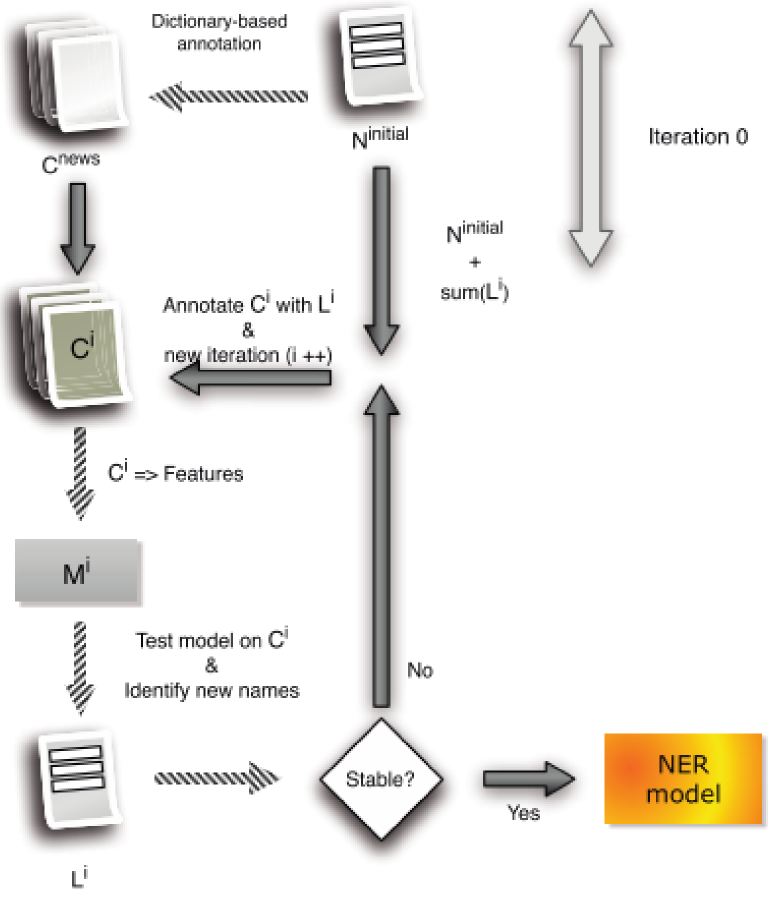
Dictionary-based annotation
C^{news}
$N^{initial}$
Iteration 0
$N^{initial}$ + sum(L^i)
Annotate C^i with L^i & new iteration (i ++)
C^i
C^i => Features
M^i
Test model on C^i & Identify new names
No
Stable?
Yes
NER model
L^i

Figure 5.1: Bootstrapping method. From [TSO11].

Name	Name extraction	Train	Test
$NewsExperiment$	$News_{total}$	$News_{partial}$	$HAREM_{second\_per}$, $News_{test\_per}$
$WikipediaExperiment$	$Wiki_{raw}$	$Wiki_{partial}$	$WikiNER_{per}$

Table 5.1: Bootstrapping experiments details. Description of the datasets available in chapter 3.

5.1 Data

The initial objective for the bootstrapping experiments was to use the same dataset that was used by Teixeira *et.al.* [TSO11], but the textual data was no longer available. As a consequence, some adjustments to the original plan had to be made. Two datasets were gathered to be used in bootstrapping experiments: one based on Portuguese news articles, just like the work of Teixeira *et.al.*; the other based on the Portuguese Wikipedia dump of April 1, 2018. Further details about the data used in the bootstrapping experiments is available in Chapter 3.

The textual data chosen to perform the bootstrapping experiments belongs to two different textual genres, so two experimental setups were created to test the bootstrapping method. Details about these two setups can be seen in Table 5.1. In the $NewsExperiment$ two test datasets are used, a previously annotated dataset, $HAREM_{second\_per}$, and a very small dataset that was annotated by hand, $News_{test\_per}$. The $News_{test\_per}$ dataset was created so that the model could be evaluated using textual data from the same time period, therefore removing the age factor from the equation. As for the $WikipediaExperiment$ only one test dataset is used, $WikiNER_{per}$, this dataset is much larger than the test datasets used in the $NewsExperiment$.

The major difference between the $NewsExperiment$ and the $WikipediaExperiment$ is the the textual genre used in training and testing the models. One of the experiments uses news articles as a data source while the other uses Wikipedia articles.

5.2 Experimental setup

The strategy to obtain the initial name list is the same for both datasets: identify a common pattern in which names of people appear with 100% precision, model a regular expression to capture the name of the person and finally extract all names from the text.

The pattern identified is vaguely the same for both news data and Wikipedia data, and follows a structure of $\langle[CapitalizedSequence],[ergonym]\rangle$ just like Teixeira *et.al.* did in 2011 [TSO11]. Some examples of words present in the ergonym list are: *presidente*, *jogador*, *vocalista*, *pai*, *marido*, *mulher*.

This pattern proved to work fine for news articles. However, to obtain the initial name list from Wikipedia text it was necessary to further restrict the pattern. Previous possible words are introduced into the pattern, modifying the initial name pattern for Wikipedia to: $\langle[possiblepreviousword]$ $[CapitalizedSequence],[ergonym]\rangle$. Possible previous words include mostly words that refer to the nationality of the person in question, like: *brasileiro*, *chileno*, *argentino*, *inglesa*.

Both the possible previous word list and the ergonym list are preprocessed to include the original lower-cased word and the respective word with the first letter capitalized. The full list of ergonyms and possible previous words is present in appendix A. In order to fully guarantee that only correct names are present in the initial name list, some further conditions were added:

1. The capitalized sequence length must be at least 2, that is, single word names are excluded.
2. For a name to be included in the initial name list it must occur at least N times for the in the name extraction dataset. $N = 3$ for the $NewsExperiment$ and $N = 2$ for the $WikipediaExperiment$.

After obtaining the initial name list it is necessary to annotate the training dataset, which consists of just raw textual data with no annotations. The initial name list is sorted by length so that the names with the most number of words are processed first. When the complete training dataset is generated, it can be interpreted as an annotated dataset. In truth, it is only partially annotated, since many names do not follow the pattern used to obtain the initial list of names and therefore have no named entity tag associated.

A model is trained using the annotated train set. This newly trained model is then tested using a standard annotated dataset. Before moving on to the next bootstrapping iteration it is necessary to update the name list, which is done by annotating the train set using the trained model. The new names obtained are added to the name list and will be used in the following bootstrapping iteration. An example of updating the name list is illustrated in Figure 5.2, at iteration 1 the name list used to annotate the training data contains only two names: *Bernardo Silva* and *Gonçalo Guedes*. After training the model and re-annotating the train textual data a new name is detected, *Bruno Fernandes*, this new name is added to the name list used in the following iteration where these operations are repeated.

After some initial experimentation with different networks, the $BiLSTM\_CNN_{Network}$ was chosen to be used in the bootstrapping experiments as it was the best performing model in the preliminary experimentations. Further details on the architecture of the $BiLSTM\_CNN_{Network}$ are available in section 4.2.4. The PyTorch implementation of Xuezhe Ma[1] was modified to perform the bootstrapping process.

The hyper-parameters used in both the News and Wikipedia experiments are available in Table 5.2. The pre-trained word embeddings used were $Wang2Vec_{64D}$, more details about the pre-trained embeddings are available in section 4.1.

5.3 Results

The results gathered with the $NewsExperiment$ setup can be compared with the results obtained by Teixeira *et.al.* [TSO11]. Teixeira *et.al.* also use textual data from news articles and test the trained models using the $HAREM_{second\_per}$. One important difference is the age of the articles used in the bootstrapping process. They use news articles from the year 2011 while the experiments performed in this work use news articles from 2018. This difference in age can have a

[1]https://github.com/XuezheMax/NeuroNLP2

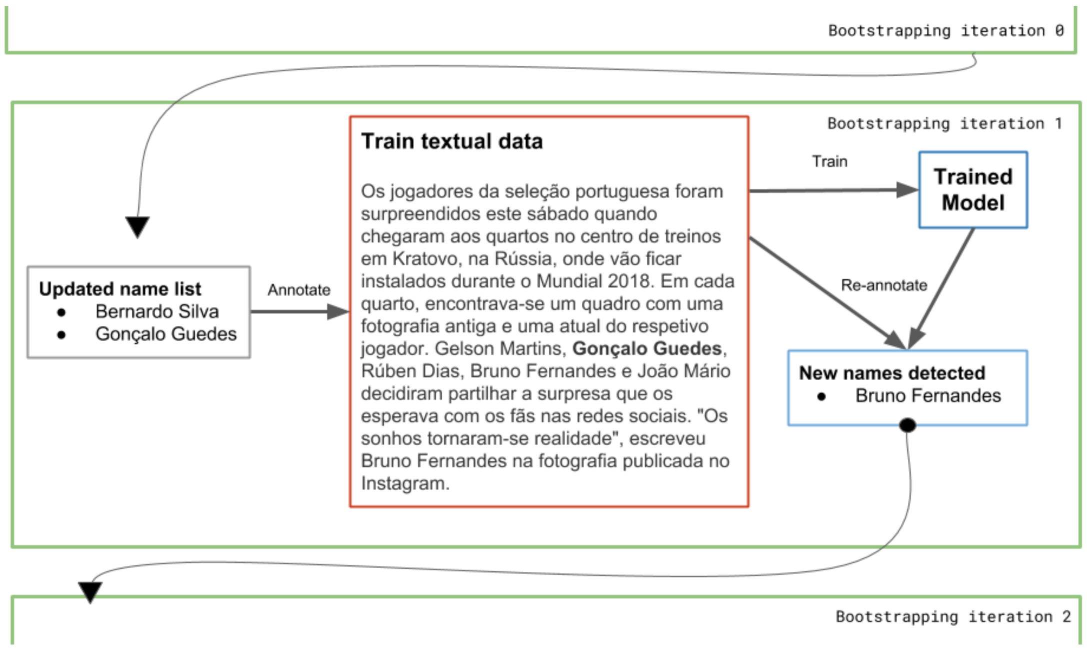
Bootstrapping iteration 0
Bootstrapping iteration 1
Updated name list
• Bernardo Silva
• Gonçalo Guedes
Annotate
Train textual data
Os jogadores da seleção portuguesa foram surpreendidos este sábado quando chegaram aos quartos no centro de treinos em Kratovo, na Rússia, onde vão ficar instalados durante o Mundial 2018. Em cada quarto, encontrava-se um quadro com uma fotografia antiga e uma atual do respetivo jogador. Gelson Martins, **Gonçalo Guedes**, Rúben Dias, Bruno Fernandes e João Mário decidiram partilhar a surpresa que os esperava com os fãs nas redes sociais. "Os sonhos tornaram-se realidade", escreveu Bruno Fernandes na fotografia publicada no Instagram.
Train
Trained Model
Re-annotate
New names detected
• Bruno Fernandes
Bootstrapping iteration 2

Figure 5.2: Running example of the bootstrapping method.

Parameter Name	Value
Batch size	16
Decay rate	0.05
Initial learning rate	0.01
LSTM hidden size	256
LSTM layers	1
Char embedding dimension	30
Number of CNN filters	30
CNN window size	3
CNN stride	1
Dropout	0.5

Table 5.2: Hyper parameters of $BiLSTM\_CNN_{Network}$ used for the bootstrapping process.

Parameter	Teixeira *et.al.*	*NewsExperiment*	*WikipediaExperiment*
Initial names	2 450	427	2 711
Docs to extract names from	500 000	57 000	892 834
Year	2011	2017 & 2018	2018
Number of tokens in training set	–	6 775 660	7 000 000

Table 5.3: Data comparison with the work of Teixeira *et.al.* [TSO11].

serious impact in the performance of the NER models, as was demonstrated by Teixeira *et.al.*. The larger age difference in comparison with the data used by Teixeira *et.al.* is expected to affect the bootstrapping performance results negatively.

Another significant difference between the *NewsExperiment* setup and the work of Teixeira *et.al.* is the difference in size of the datasets involved in the bootstrapping process. These differences are highlighted in Table 5.3. Included in Appendix A are two tables, A.1 and A.2, that list the performance measures per bootstrapping cycle for both the *NewsExperiment* and *WikipediaExperiment*.

Analysing the performance results present in Table 5.4, it is clear that the bootstrapping process is not fit to handle Wikipedia articles, at least not using the same parameters as the ones used for the *NewsExperiment*. The reasons behind the poor results in the *WikipediaExperiment* could be related to the textual genre, Portuguese Wikipedia articles contain more foreign names of people than Portuguese media articles and the context in which names appear within the article is different in the two textual genres. Furthermore the test set for the *WikipediaExperiment*, $WikiNER_{per}$ is a much larger dataset than the test set used for the *NewsExperiment*.

Since the results from the *WikipediaExperiment* only proved that the described bootstrapping process is not compatible with that textual genre, the discussion will focus on the results of the *NewsExperiment*.

It is possible to view the evolution of the performance scores in both the $HAREM_{second\_per}$ and $News_{test\_per}$ datasets for each of the bootstrapping iterations in Figure 5.3 and Figure 5.4. The evolution of the number of new names added to the initial name set for each bootstrapping iteration is illustrated in Figure 5.5. It is clear that despite starting with a much lower number of initial names and the training set being smaller than the one used in the work of Teixeira *et.al.* the

Experiment	Total Iterations	F1	Precision	Recall	Avg. # new names
NewsExperiment	29	25.3	71.23	15.38	153
WikipediaExperiment	20	0.49	5.59	0.26	1.3
Teixeira *et.al.*	12	69	90	56	233

Table 5.4: Bootstrapping result comparison. Score measures correspond to the best models for all bootstrapping iterations. *NewsExperiment* and the work of Teixeira *et.al.* is tested with the $HAREM_{second\_per}$ dataset while *WikipediaExperiment* is tested with the $WikiNER_{per}$ dataset.

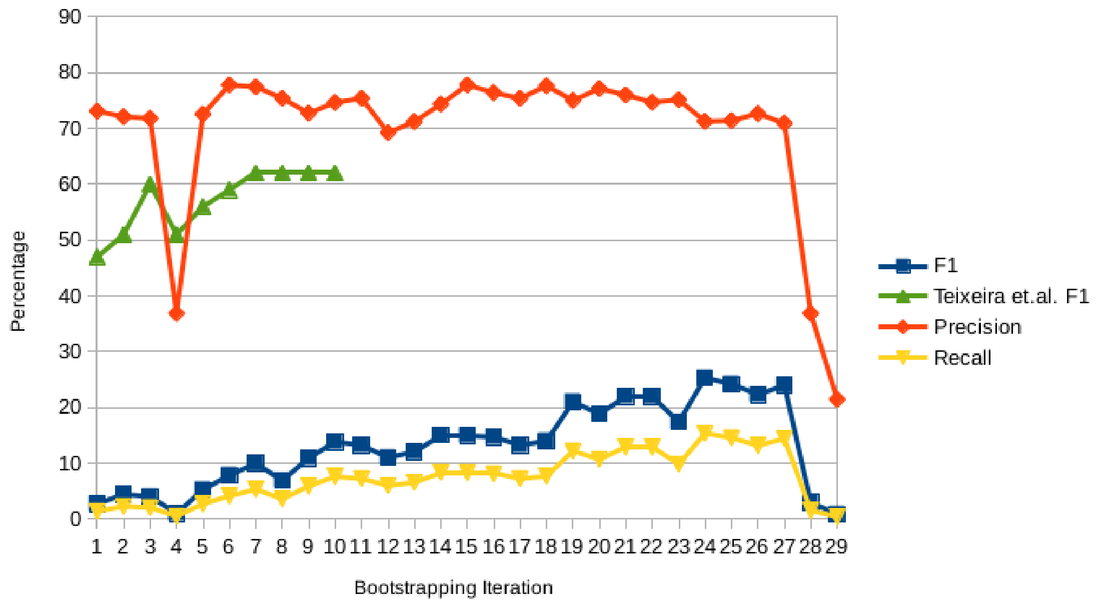

Figure 5.3: $HAREM_{second\_per}$ test scores for the *NewsExperiment* setup compared to the work of Teixeira *et.al.* [TSO11].

number of total names in the *NewsExperiment* after finishing the bootstrapping process is similar.

Looking at Figures 5.3 and 5.4 it is clear that the difference in the F1 score is due to differences in recall and not in precision. Precision values are very similar for both the $HAREM_{second\_per}$ and $News_{test\_per}$ datasets. The observed difference in recall may be due to the age difference between the train and test datasets. $HAREM_{second\_per}$ is a collection of texts from the late 90s, while the $News_{test\_per}$ dataset is made up of a small number of articles from the same time frame as the train data.

The performance measures for both test datasets are correlated and vary in the same way throughout the bootstrapping iterations. A drop in performance also means less new names are discovered at that iteration.

Comparing the results reported by Teixeira *et.al.* with the results obtained in the *NewsExperiment*, it seems that using DL in combination with bootstrapping does not provide good results. However the work of Teixeira *et.al.* can not be directly compared with the *NewsExperiment*. Teixeira *et.al.* use larger datasets from a different time frame, do not describe the preprocessing of the datasets and do not provide the ergonym list used. These differences are a source of inconsistency in the experiment setup and prohibit a fair comparison of results.

The hyper-parameters used in the bootstrapping experiments are the same as the ones used in the experiments with annotated datasets in Chapter 4. The fact that the bootstrapping process only handles one named entity category might affect the optimal hyper-parameters. Hyper-parameter tuning was not performed due to the fact that bootstrapping experiments take a long time to run, meaning results could not be obtained in a valid time-line.

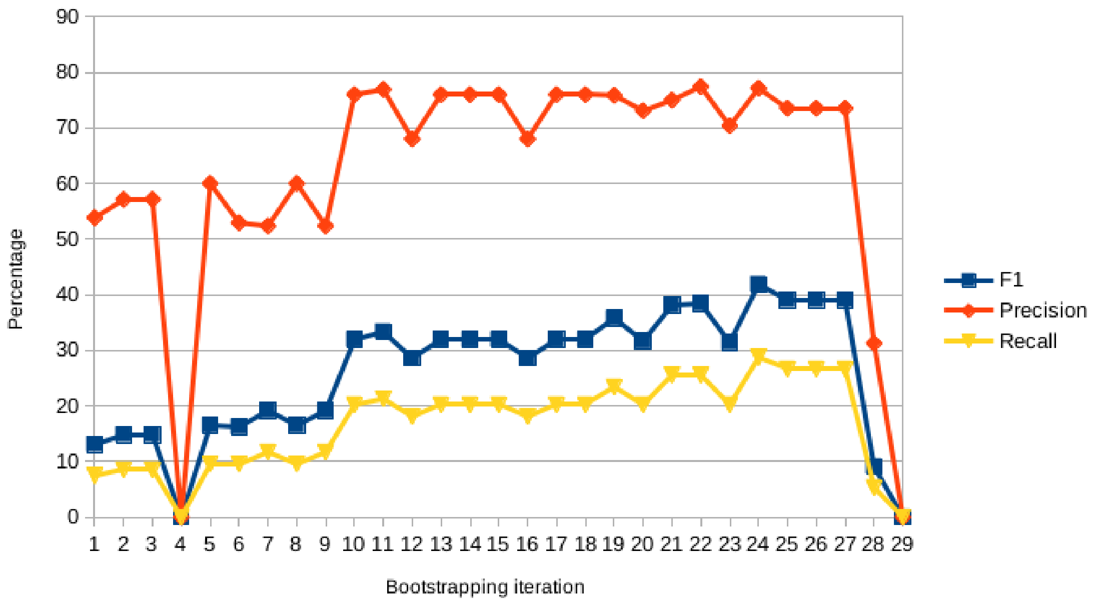

Figure 5.4: *News*$_{test\_per}$ test scores for the *NewsExperiment* setup.

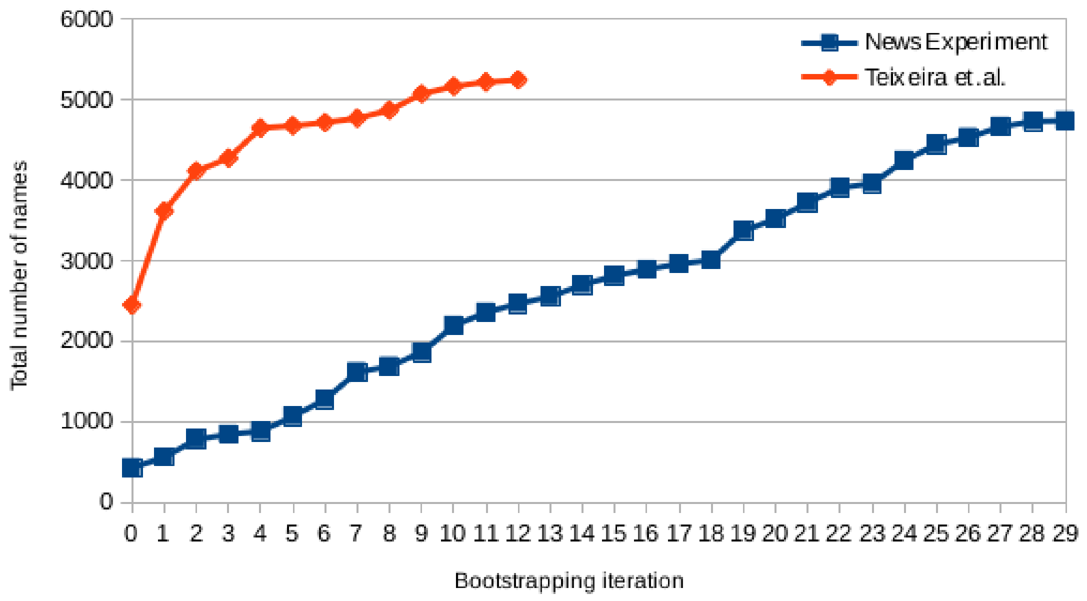

Figure 5.5: Total number of names at each bootstrapping iteration for the *NewsExperiment* setup compared with the work of Teixeira *et.al.* [TSO11].

Chapter 6

Conclusions and Future Work

This chapter focuses on analysing the work done, the discussion is split into three sections. Starting with objective completion, where the work done is compared with the initial outlined objectives. Followed by the future work section where possible improvements and follow-up work are discussed. The final section reflects on what were the lessons learned throughout the multiple stages.

The main contributions of this work are the train and test of multiple state-of-the-art deep learning NER architectures with Portuguese texts. Another highlight of this work is the fact that both annotated data and non-annotated data are explored to create NER models.

6.1 Objective Completion

To evaluate objective completion it is necessary to look back at the objective (Section 1.3) and research guidelines (Section 1.4) and see to what extent did the work done follow them. The research guidelines list the tasks required to correctly evaluate the viability of applying DL architectures to NER in Portuguese texts.

State of the art deep learning architectures for NER proved to be adequate for Portuguese datasets. The drop in performance when compared to the English results can be attributed to multiple factors, as discussed in Section 4.6, but not to the architectures themselves or to the Portuguese language specifically.

This work started by collecting and preprocessing various textual resources and NER datasets (Chapter 3) and the next steps were selecting and testing different deep learning NER architectures. The criteria for selecting the different architectures was the performance reported in the CoNLL-2003 English NER dataset: all the chosen architectures were at some point in time considered the state-of-the-art for NER in the English language. All the selected architectures were trained and tested with Portuguese datasets. Details about the architectures, the experiment setup and results are all available in Chapter 4.

Comparisons between English corpora and Portuguese corpora are available both is Chapter 2 and in Section 4.6. As highlighted there, besides being smaller in size, Portuguese datasets have some characteristics, like the average number of tokens per sentence, that might be one of the reasons for the observed differences in performance.

In the bootstrapping experiments (Chapter 5) NER is tested with different textual genres: news articles and Wikipedia articles. The results revealed some challenges linked to textual genre: models trained with Wikipedia data performed much worse than models trained with News articles.

The various challenges of working with DL architectures are scattered over this document. Among them are the computational resources required and the long training times. From all the difficulties encountered, only one can be directly linked to the Portuguese language, which is the low amount of textual resources freely available.

It is possible to say that the objective was achieved: multiple experiments with DL architectures for NER in the Portuguese language were ran. Results obtained were critically analysed and conclusions drawn.

6.2 Future Work

There are a multitude of ways to improve and develop on the work done. Most of the improvements have to do with exploring more architectures for both the NER models and pre-training embeddings but also including more data into the training process.

In addition to exploring different DL architectures applied to NER like the work of Gillick *et.al.* [GBVS15] or the work of Yang *et.al.* [YZD17], it is also important to tune the hyper-parameters of the architectures used to better fit the Portuguese datasets. Hyper-parameter tuning is a time consuming task that requires training and testing multiple models with different hyper-parameter combinations. The better performing combination is then used to run the full experiments. A more detailed discussion on the reasons behind not running hyper-parameters tuning is present in Section 4.3.

Some of the architectures tested use techniques, like dropout, that introduce randomness in the training process. Randomness in the training process means that models trained with the same architecture and parameters perform differently in different runs. To guarantee that results are not biased due to an above average model being tested, it is important to re-run the same training process multiple times for the same architecture and report the average and standard deviation.

Further work could include exploring other NLP tasks related to NER such as entity linking or co-reference resolution and performing the same analysis of the applicability of DL methods for the Portuguese language. Named entity linking is a follow-up step to NER and aims to link entity mentions within the same document or in other resources [DMR^{+}15]. The target of co-reference resolution is to find words in the given text that refer to the same thing or person [DM14].

In a broader sense, future work for NER in Portuguese should include the creation of a large annotated dataset so that state of the art models, such as deep learning architectures, can be explored in the future. A large dataset that follows the standard format adopted by the majority of the NLP community and shares the same evaluation strategy would be a great contribution. This new dataset would help Portuguese NER research to stay up to date with all the NER developments to come.

Following the same logic, the bootstrapping experiments that use non annotated data would also benefit from larger amounts of textual data being available. Creating an easy and free way to obtain large amounts of Portuguese textual data would benefit the research on not only NER but Portuguese NLP in general. Large amounts of textual data have multiple applications, including: training NER models using techniques like bootstrapping and training better word embeddings. The use of embeddings trained with large amounts of data that have repeatedly shown to improve the performance of multiple NLP tasks.

6.3 Lessons Learned

After going through the multiple stages that made up this work many lessons were learned. In this section the most relevant ones are highlighted. The first lesson learned was right at the start of this work, when different architectures were being analysed and understood. Important aspects and critical details are sometimes omitted from the paper that introduces the architecture. Aspects like preprocessing of data or parameter initialization are critical details required to replicate the architecture and eventual results but are not always present.

Time is a valuable resource, and developing work that is constrained in time is a challenge. No matter how long the deadline is, at the end there are always some more experiments to be run and ideas to explore. There must be a point where the decision is made to stop experimenting and start reporting the results obtained, otherwise the end result would just be a multitude of unprocessed results with no conclusions drawn about all the different experiments ran.

Obtaining raw unprocessed textual data is not always easy. Even though large amounts of texts are available to consume for free in news websites, blog posts or forums, not all of these platforms allow the textual data to be extracted in an easy way. Creating large, freely available and easily distributed textual corpora for the Portuguese language would greatly benefit NLP developments.

Different datasets can have a different number of named entity categories and even include sub-categories, some support overlapping named entities or multi-class named entities. These different characteristics of the datasets lead to different evaluation methods, which means evaluation metrics for the different models can not be compared in different datasets. Having a standard dataset format, class set and evaluation metric for NER models that is adopted by the international NLP community would promote knowledge sharing between research in different languages and help the progress of NER in all languages.

References

[ABP$^+$16] Vinayak Athavale, Shreenivas Bharadwaj, Monik Pamecha, Ameya Prabhu, and Manish Shrivastava. Towards Deep Learning in Hindi NER: An approach to tackle the Labelled Data Scarcity. *CoRR*, abs/1610.0, oct 2016.

[AHFB17] Ahmed Sultan Al-Hegami, Ameen Mohammed Farea Othman, and Fuad Tarbosh Bagash. A Biomedical Named Entity Recognition Using Machine Learning Classifiers and Rich Feature Set. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY*, 17(1):170–176, 2017.

[BGJM16] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. *CoRR*, abs/1607.0, 2016.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 1 edition, 2009.

[BON03] Oliver Bender, Franz Josef Och, and Hermann Ney. Maximum entropy models for named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 -*, volume 4, pages 148–151, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[Car12] Nuno Cardoso. Rembrandt-a named-entity recognition framework. In Nicoletta Calzolari Piperidis, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, pages 1240–1243. 2012.

[CF13] Flavio Massimiliano Cecchini and Elisabetta Fersini. Named entity recognition using conditional random fields with non-local relational constraints. *CoRR*, abs/1310.1, oct 2013.

[CM12] Angel X Chang and Christopher D Manning. SUTime: A library for recognizing and normalizing time expressions. *Lrec*, (iii):3735–3740, 2012.

[CMP02] X Carreras, L Marquez, and L Padró. Named entity extraction using adaboost. In *proceeding of the 6th conference on Natural language learning - COLING-02*, volume 20, pages 0–3, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[CN03] Hai Leong Chieu and Hwee Tou Ng. Named entity recognition with a maximum entropy approach. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 -*, volume 4, pages 160–163, 2003.

[CN15] Jason P. C. Chiu and Eric Nichols. Named Entity Recognition with Bidirectional LSTM-CNNs. *CoRR*, abs/1511.0, nov 2015.

[CW08] Ronan Collobert and Jason Weston. A unified architecture for natural language processing. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 160–167, New York, New York, USA, 2008. ACM Press.

[CWB$^+$11] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (almost) from Scratch. *CoRR*, abs/1103.0, mar 2011.

[dABV13] Sandra Collovini de Abreu, Tiago Luis Bonamigo, and Renata Vieira. A review on Relation Extraction with an eye on Portuguese. *Journal of the Brazilian Computer Society*, 19(4):553–571, nov 2013.

[dAV13] Daniela O. F. do Amaral and Renata Vieira. O Reconhecimento de Entidades Nomeadas por meio de Conditional Random Fields para a Língua Portuguesa. In *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology*, pages 59–68, 2013.

[DLCT15] Hong Jie Dai, Po Ting Lai, Yung Chun Chang, and Richard Tzong Han Tsai. Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization. *Journal of Cheminformatics*, 7:S14, 2015.

[DM14] Rahul Dudhabaware and Mangala Madankar. Review on Natural Language Processing Tasks for Text Documents. In *IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–5. IEEE, dec 2014.

[DMR$^+$15] Leon Derczynski, Diana Maynard, Giuseppe Rizzo, Marieke Van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak, and Kalina Bontcheva. Analysis of named entity recognition and linking for tweets. *Information Processing and Management*, 51(2):32–49, 2015.

[dSG15] Cícero Nogueira dos Santos and Victor Guimarães. Boosting Named Entity Recognition with Neural Character Embeddings. *CoRR*, abs/1505.0, may 2015.

[EB10] Asif Ekbal and Sivaji Bandyopadhyay. Named Entity Recognition using Support Vector Machine: A Language Independent Approach. *World Academy of Science, Engineering and Technology*, 39(September):548–563, 2010.

[FGM05] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics - ACL '05*, pages 363–370, 2005.

[FMS$^+$10] Cláudia Freitas, Cristina Mota, Diana Santos, Hugo Gonçalo Oliveira, and Paula Carvalho. Second HAREM : Advancing the State of the Art of Named Entity Recognition in Portuguese. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, number 3, pages 3630–3637, 2010.

[For73] G. David Forney. The Viterbi Algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[GBC16] I Goodfellow, Y Bengio, and A Courville. *Deep learning*, volume 22. MIT Press, 2016.

[GBVS15] Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. Multilingual Language Processing From Bytes. *CoRR*, abs/1512.0, nov 2015.

[GG15] Marcos Garcia and Pablo Gamallo. Yet another suite of multilingual NLP tools. In *Communications in Computer and Information Science*, volume 563, pages 65–75, 2015.

[GS96] Ralph Grishman and Beth Sundheim. Message Understanding Conference-6: A Brief History. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, volume 1, pages 466—-471, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[Hak13] Jörg Hakenberg. Named Entity Recognition. *Encyclopedia of Systems Biology*, pages 1487–1488, 2013.

[HFS$^+$17] Nathan Hartmann, Erick Fonseca, Christopher Shulby, Marcos Treviso, Jessica Rodrigues, and Sandra Aluisio. Portuguese Word Embeddings: Evaluating on Word Analogies and Natural Language Tasks. *CoRR*, abs/1708.0, aug 2017.

[HMP$^+$06] Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. OntoNotes: the 90% solution, 2006.

[HXY15] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. *CoRR*, abs/1508.0, aug 2015.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[LBS$^+$16] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural Architectures for Named Entity Recognition. *CoRR*, abs/1603.0, mar 2016.

[LDBT15] Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. Two/Too Simple Adaptations of Word2Vec for Syntax Problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304, 2015.

[LMP01] John Lafferty, Andrew McCallum, and Fernando C N Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*, 8(June):282–289, 2001.

[LZWZ11] Xiaohua Liu, Shaodian Zhang, Furu Wei, and Ming Zhou. Recognizing Named Entities in Tweets. *In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, 1(2008):359–367, 2011.

[MCCD13] Tomas Mikolov, Greg Corrado, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3, 2013.

[MD07] Ruy Luiz Milidiú and Julio Cesar Duarte. Machine Learning Algorithms for Portuguese Named Entity Recognition. *Intel. Artif.*, 11(36):67–75, 2007.

[MFP00] Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598, 2000.

[MH16] Xuezhe Ma and Eduard Hovy. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. *CoRR*, abs/1603.0, 2016.

[Mik13] Tomas Mikolov. Learning Representations of Text using Neural Networks (Slides). *NIPS Deep Learning Workshop*, abs/1310.4:1–31, 2013.

[MSB$^+$14] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. The {Stanford} {CoreNLP} Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[MUSC$^+$13] Mónica Marrero, Julián Urbano, Sonia Sánchez-Cuadrado, Jorge Morato, and Juan Miguel Gómez-Berbís. Named Entity Recognition: Fallacies, challenges and opportunities. *Computer Standards and Interfaces*, 35(5):482–489, sep 2013.

[Nad07] David Nadeau. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[Nam17] Mahdi Namazifar. Named Entity Sequence Classification. *ArXiv e-prints*, 2017.

[NRR$^+$13] Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R. Curran. Learning multilingual named entity recognition from Wikipedia. *Artificial Intelligence*, 194:151–175, 2013.

[Pir17] André Ricardo Oliveira Pires. *[THESIS] Named entity extraction from Portuguese web text.* PhD thesis, Faculty of Engineering of University of Porto, 2017.

[PKM14] Alexandre Passos, Vineet Kumar, and Andrew McCallum. Lexicon Infused Phrase Embeddings for Named Entity Resolution. *CoRR*, abs/1404.5, apr 2014.

[PKPS00] Georgios Paliouras, Vangelis Karkaletsis, Georgios Petasis, and Constantine D Spyropoulos. Learning Decision Trees for Named-Entity Recognition and Classification. In *ECAI Workshop on Machine Learning for Information Extraction*. 2000.

[PS12] Lluis Padró and Evgeny Stanilovsky. FreeLing 3.0: Towards Wider Multilinguality. In *Proceedings of the Language Resources and Evaluation Conference (LREC 2012)*, pages 2473–2479, 2012.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[RMD13] Delip Rao, Paul McNamee, and Mark Dredze. Entity Linking: Finding Extracted Entities in a Knowledge Base. *Multi-source, Multilingual Information Extraction and Summarization*, pages 93–115, 2013.

[RR09] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. *CoNLL: Conference on Computational Natural Language Learning*, 30(June):147–155, 2009.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.0, 2016.

[Sar06] Luís Sarmento. SIEMÊS - A named-entity recognizer for Portuguese relying on similarity rules. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3960 LNAI, pages 90–99. Springer, Berlin, Heidelberg, 2006.

[SC06] Diana Santos and Nuno Cardoso. A Golden Resource for Named Entity Recognition in Portuguese. In *Proceedings of the 7th International Workshop, PROPOR 2006*, pages 69–79, 2006.

[SD03] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 -*, volume 4, pages 142–147, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[SFK06] György Szarvas, Richárd Farkas, and András Kocsor. A Multilingual Named Entity Recognition System Using Boosting and C4 . 5 Decision Tree Learning Algorithms. In *Discovery Science*, pages 267–278. Springer, Berlin, Heidelberg, 2006.

[SHK$^+$14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[SM13] Richard Socher and Christopher Manning. Deep Learning for NLP. *HLT-NAACL Tutorials*, pages 1–204, 2013.

[SPC06] Luís Sarmento, Ana Sofia Pinto, and Luís Cabral. REPENTINO - A wide-scope gazetteer for entity recognition in portuguese. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3960 LNAI, pages 31–40. Springer Verlag, 2006.

[SS15] Amarappa S and Sathyanarayana S.V. Kannada Named Entity Recognition and Classification (NERC) Based on Multinomial Naïve Bayes (MNB) Classifier. *International Journal on Natural Language Computing*, 4(4):39–52, 2015.

[SSCV06] Diana Santos, Nuno Seco, Nuno Cardoso, and Rui Vilela. HAREM: An Advanced NER Evaluation Contest for Portuguese. In *Proceedings of the 5th International Conference on Language Resources and Evaluation, LREC'2006*, pages 1986—-1991, 2006.

[SSN02] Satoshi Sekine, Kiyoshi Sudo, and Chikashi Nobata. Extended Named Entity Hierarchy. In *Language Resources And Evaluation Conference*, pages 1818–1824, 2002.

[Ste12] Rosa Stern. A Joint Named Entity Recognition and Entity Linking System. In *Proceedings of the Workshop on Innovative Hybrid Approaches to the Processing of Textual Data*, pages 52–60, 2012.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. *CoRR*, abs/1409.3, sep 2014.

[SWH15] Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2015.

[SZ14] CD Santos and B Zadrozny. Learning Character-level Representations for Part-of-Speech Tagging. *Proceedings of the 31st International Conference on Machine Learning*, ICML-14(2011):1818–1826, 2014.

[SZH17] Shengli Song, Nan Zhang, and Haitao Huang. Named entity recognition based on conditional random fields, sep 2017.

[TC02] Koichi Takeuchi and Nigel Collier. Use of Support Vector Machines in Extended Named Entity Recognition. In *Proceedings of the 6th Conference on Natural Language Learning - Volume 20*, COLING-02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[Tjo02] Erik F. Tjong Kim Sang. Introduction to the CoNLL-2002 shared task. In *proceeding of the 6th conference on Natural language learning - COLING-02*, volume 20, pages 1–4, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[TRM08] BT Todorovic, SR Rancic, and IM Markovic. Named entity recognition and classification using context Hidden Markov Model. In *Neural Network Applications in Electrical Engineering, 2008. NEUREL 2008. 9th Symposium on*, number 1, pages 43–46. IEEE, sep 2008.

[TSO11] Jorge Teixeira, Luís Sarmento, and Eugénio Oliveira. A bootstrapping approach for training a ner with conditional random fields. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7026 LNAI, pages 664–678, 2011.

[VR08] Renata Vieira and Sandro Rigo. Reconhecimento automático de relações entre entidades mencionadas em textos de língua portuguesa. pages 1–11, 2008.

[WPR$^+$12] Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Nianwen Xue, Martha Palmer, Jena D Hwang, Claire Bonial, Jinho Choi, Aous Mansouri, Maha Foster, Abdel-Aati Hawwary, Mitchell Marcus, Ann Taylor, Craig Greenberg, Eduard Hovy, Robert Belvin, and Ann Houston. OntoNotes Release 5.0. Technical report, 2012.

[YZD17] Jie Yang, Yue Zhang, and Fei Dong. Neural Reranking for Named Entity Recognition. *CoRR*, abs/1707.0, jul 2017.

[ZS01] GuoDong Zhou and Jian Su. Named entity recognition using an HMM-based chunk tagger. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, page 473, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

Appendix A

Bootstrapping experiment details

The full list of ergonyms used to detect the initial list of names:

```
['presidente', 'jogador', 'treinador', 'deputado', 'arcebispo', 'pastor',
'fundador', 'companheiro', 'licenciado', 'residente', 'dinamizador',
'especialista', 'candidato', 'jornalista', 'comerciante', 'autor',
'conselheiro', 'professor', 'ministro', 'comandante', 'chefe',
'guitarrista', 'atriz', 'actor', 'vocalista', 'cantor', 'cantora',
'produtor', 'físico', 'descobridor', 'piloto',
'almirante', 'encenador', 'arquiteto', 'governador', 'cardeal', 'bispo',
'músico', 'irmão', 'irmã', 'filho', 'filha', 'pai', 'mãe', 'avô', 'avó',
'tio', 'tia', 'primo', 'prima', 'marido', 'mulher']
```

News specific ergonym list:

```
['ex-líder', 'especializado', 'especializada', 'apelou', 'desafiou',
'militante', 'administrador', 'vice-ministro', 'presidiu', 'homenageou',
'criou', 'alertou', 'mostrou']
```

The full list of possible previous words used to detect the initial list of names:

```
['por', 'como', 'segundo', 'iorquino', 'canadense',
'alemão', 'brasileiro', 'brasileira', 'português', 'portuguesa',
'estadunidense', 'americana', 'americano', 'coreano', 'francês',
'espanhol', 'espanhola', 'fracesa', 'alemão', 'alemã', 'inglês',
'inglesa', 'britânico', 'britânica', 'chileno', 'chilena', 'argentino',
'peruano', 'peruana', 'ucraniano', 'ucraniana', 'islandesa', 'islandês']
```

Iteration	$HAREM_{second\_per}$			$News_{test\_per}$			NewNames
	F1	Precision	Recall	F1	Precision	Recall	
1	2.76	73.08	1.41	13.08	53.85	7.45	132
2	4.44	72.09	2.29	14.81	57.14	8.51	223
3	4.03	71.79	2.07	14.81	57.14	8.51	60
4	1.02	36.84	0.52	0	0	0	38
5	5.27	72.55	2.74	16.51	60	9.57	186
6	7.87	77.78	4.14	16.22	52.94	9.57	213
7	9.97	77.42	5.33	19.13	52.38	11.7	339
8	6.92	75.38	3.62	16.51	60	9.57	70
9	10.94	72.73	5.92	19.13	52.38	11.7	173
10	13.83	74.64	7.62	31.93	76	20.21	333
11	13.23	75.38	7.25	33.33	76.92	21.28	165
12	11.03	69.23	5.99	28.57	68	18.09	108
13	12.05	71.2	6.58	31.93	76	20.21	87
14	15.03	74.34	8.36	31.93	76	20.21	146
15	14.97	**77.78**	8.28	31.93	76	20.21	112
16	14.71	76.39	8.14	28.57	68	18.09	80
17	13.23	75.38	7.25	31.93	76	20.21	68
18	14	77.61	7.69	31.93	76	20.21	48
19	20.99	75	12.2	35.77	75.86	23.4	365
20	18.83	77.13	10.72	31.67	73.08	20.21	147
21	22.01	75.98	12.87	38.1	75	25.53	202
22	21.96	74.68	12.87	38.4	**77.42**	25.53	186
23	17.4	75.14	9.84	31.4	70.37	20.21	50
24	**25.3**	71.23	**15.38**	**41.86**	77.14	**28.72**	287
25	24.2	71.38	14.57	39.06	73.53	26.6	198
26	22.29	72.65	13.17	39.06	73.53	26.6	88
27	23.97	70.91	14.42	39.06	73.53	26.6	134
28	2.98	36.84	1.55	9.09	31.25	5.32	64
29	0.87	21.43	0.44	0	0	0	8

Table A.1: News experiment bootstrapping results.

Iteration	F1	Precision	Recall	NewNames
1	0.27	3.14	0.14	1
2	0.36	4.28	0.19	1
3	0.04	0.52	0.02	0
4	0.21	2.45	0.11	5
5	0.43	5	0.23	0
6	0.09	1.06	0.05	0
7	0.04	0.45	0.02	1
8	0.36	4.25	0.19	0
9	0.04	0.45	0.02	4
10	0.49	5.57	0.25	0
11	0.48	5.53	0.25	0
12	0.19	2.32	0.1	1
13	0.31	3.64	0.16	1
14	0.17	2.05	0.09	0
15	0.06	0.73	0.03	7
16	0.41	4.72	0.21	1
17	0.16	1.99	0.09	1
18	0.36	4.17	0.19	0
19	0.04	0.51	0.02	0
20	0.49	5.59	0.26	3

Table A.2: Wikipedia experiment bootstrapping results. Score results in the $WikiNER_{per}$ dataset.