Automated extraction of research questions and answers from medical publications

Karatzia Anastasia
Supervisor: Prof. Ioannis Vlahavas

Thessaloniki, March 2020
To my family,
Αυτοματοποιημένη εξαγωγή ερευνητικών ερωτήσεων και απαντήσεων από ιατρικές δημοσιεύσεις.

Περίληψη
Η ιατρική βασισμένη σε τεκμηρίωση είναι μια συστηματική προσέγγιση της επίλυσης ιατρικών προβλημάτων, η οποία επιτρέπει την ενσωμάτωση των βέλτιστων διαθέσιμων στοιχείων έρευνας με την κλινική εμπειρογνωμοσύνη. Οι γιατροί καλούνται να ακολουθήσουν ένα βασικό μοντέλο, το οποίο ονομάζεται PICO μοντέλο, προκειμένου να ασκήσουν ιατρική που βασίζεται στην τεκμηρίωση. Στο πλαίσιο της παρουσίας διπλωματικής εργασίας, προτείνεται μια ποικιλία νευρονικών μοντέλων που βασίζονται σε Διπλής Κατεύθυνσης Μακράς Διάρκειας μοντέλα μνήμης, για την αναγνώριση των βασικών PICO οντοτήτων. Αυτά τα μοντέλα περιλαμβάνουν Διπλής Κατεύθυνσης Μακράς Διάρκειας δίκτυα μνήμης με επίπεδα Υποθετικών Τυχαίων Πεδίων ή Διπλής Κατεύθυνσης Μακράς Διάρκειας μοντέλα μνήμης σε συνδυασμό με επίπεδα Υποθετικών Τυχαίων Πεδίων και επίπεδα Προσοχής. Τα εν λόγω μοντέλα εκπαιδεύτηκαν σε ένα σύνολο από 5000 περιλήψεις ιατρικών κειμένων που περιγράφουν κλινικές μελέτες. Διαφορετικές αναπαραστάσεις λέξεων δοκιμάστηκαν στα μοντέλα, σε επίπεδο λέξεων, επίπεδο χαρακτήρων και επίπεδο ετικέτας τμήματος ομίλιας. Τα προκαταρκτικά αποτελέσματα αποκάλυψαν ότι το μοντέλο των νευρονικών δικτύων λειτουργεί καλά, αλλά περαιτέρω έρευνες είναι απαραίτητες για την επίτευξη αποτελεσμάτων τελευταίας τεχνολογίας.

Λέξεις Κλειδιά: Ιατρική βασισμένη στην τεκμηρίωση, Ακολουθία ετικετών, Βαθιά μάθηση, Διπλής Κατεύθυνσης Μακράς Διάρκειας μοντέλο μνήμης, Υποθετικό Τυχαίο Πεδίο.
Abstract

Evidence based medicine is a systematic approach to clinical problem solving which allows the integration of the best available research evidence with clinical expertise and patient values. Physicians are required to follow a basic model, named the PICO model in order to practice evidence based medicine. In the context of this thesis, a variety of Bidirectional Long Short – Term Memory based models is proposed for named entity recognition of the basic PICO entities. These models include Bidirectional Long Short – Term Memory networks with a Conditional Random Field layer or Bidirectional Long Short – Term Memory networks with a Conditional Random Field layer and an Attention layer. These models were trained on a corpus of 5000 richly annotated abstracts of medical articles describing clinical randomized control trials. Different word vector representations were tested in the models, including word level, character level and part – of – speech tag level. Preliminary results revealed that the neural networks model performs well but further investigations are necessary in order to achieve state-of – the – art results.

Keywords: Evidence Based Medicine, Sequence tagging, Deep learning, Bidirectional Long Short – Term Memory model, Conditional Random Field.
Acknowledgments

First of all, I would like to thank professor I. Vlahava and doctoral student N. Stylianou for their help and guidance during this thesis.

In addition, I would like to thank and dedicate this thesis to my grandfather Konstantinos, who always encouraged me to be better and to my parents for their love and support in every decision in my life. Furthermore I would like to thank my dear friend Klelia, who was my study partner on this master and Sakis for his enormous patience, help and support.

Anastasia Karatzia,
Thessaloniki 2020
## Contents

Περίληψη ............................................................................................................................... 2

Abstract ............................................................................................................................... 3

Acknowledgments ................................................................................................................. 4

List of figures ......................................................................................................................... 7

List of Tables ......................................................................................................................... 9

Chapter 1 Introduction .......................................................................................................... 11
  1.1 Motivation/Object ........................................................................................................ 11
  1.2 Course of investigation ............................................................................................. 12

Chapter 2 Natural Language Processing ........................................................................... 13
  2.1 Natural language processing tasks ............................................................................ 14
    2.1.1 Syntax tasks ....................................................................................................... 14
    2.1.2 Semantics tasks ................................................................................................ 15
    2.1.3 Discourse tasks ................................................................................................ 18
    2.1.4 Speech tasks .................................................................................................... 18
  2.2 Named Entity Recognition ......................................................................................... 19
    2.2.1 Named Entity Recognition techniques ............................................................. 19
    2.2.2 Features in Named Entity Recognition systems ................................................. 23
    2.2.3 Named Entity Recognition with Deep learning techniques ............................... 25
    2.2.4 Named Entity Recognition applications ............................................................ 29
    2.2.5 Challenges in Named Entity Recognition .......................................................... 30

Chapter 3 Evidence Based Medicine ............................................................................... 31
  3.1 The five step Evidence Based Medicine model ............................................................ 31
  3.2 Evidence Based Medicine models .............................................................................. 33
    3.2.1 PICO framework ............................................................................................... 33
    3.2.2 Other Evidence Based Medicine Frameworks .................................................... 35
  3.3 Machine learning in EBM ......................................................................................... 36

Chapter 4 Materials and Methods .................................................................................... 41
  4.1 The dataset .................................................................................................................. 41
    4.1.1 Data collection ................................................................................................... 41
    4.1.2 Data .................................................................................................................. 43
  4.2 Theoretical Background .............................................................................................. 45
4.2.1 Recurrent Neural Networks ................................................................. 45
4.2.2 Conditional Random Field theory .................................................. 50
4.2.3 Embeddings .................................................................................. 51
4.2.4 Attention mechanism ................................................................. 52
4.3 Methodology ...................................................................................... 54
  4.3.1 Preprocessing ................................................................................ 54
  4.3.2 Models ......................................................................................... 57
  4.3.3 Experimental setup and training procedure .................................. 68
  4.3.4 Evaluation and prediction .......................................................... 68
Chapter 5 Results ..................................................................................... 71
Chapter 6 Conclusions .......................................................................... 87
References .............................................................................................. 89
List of figures
Figure 1: Geometric interpretation of SVM [30].................................................................21
Figure 2: Biological neuron ..............................................................................................25
Figure 3: Threshold Logic Unit [38]................................................................................26
Figure 4: Artificial neural network [30]............................................................................27
Figure 5: Autoencoder [30]...........................................................................................28
Figure 6: Demner-Fushman and Lin’s clinical QA system architecture [50]...................36
Figure 7: Overall architecture of the QA system MEANS [59].........................................39
Figure 8: A recurrent neuron (left), unrolled through time (right). ..................................46
Figure 9: A layer of recurrent neurons (left), unrolled through time (right)......................46
Figure 10: Backpropagation through time. ......................................................................47
Figure 11: LSTM cell .......................................................................................................48
Figure 12: A Bi-directional LSTM network. .................................................................50
Figure 13: Dependencies between word vectors – GloVe. ..............................................52
Figure 14: Weights are assigned to input words at each step of the translation...............53
Figure 15: Number of tokens (x-axis) to number of sentences (y-axis) for the train set .......55
Figure 16: Number of tokens (x-axis) to number of sentences (y-axis) for the validation set ..55
Figure 17: Number of tokens (x-axis) to number of sentences (y-axis) for the test set ........56
Figure 18: BiLSTM – CRF network architecture. .............................................................58
Figure 19: BiLSTM network with word vector representations as input..........................59
Figure 20: Word level representation from character embeddings. ................................60
Figure 21: BiLSTM network with word and character vector representations as input ......61
Figure 22: BiLSTM network with word, character and POS vector representations as input .....62
Figure 23: Attention mechanism on POS embeddings. ....................................................64
Figure 24: BiLSTM network with word, character and POS attention vector representations as input ........................................................................................................65
Figure 25: Attention mechanism on the concatenation of words and character embeddings ......66
Figure 26: BiLSTM – CRF with attention layer .................................................................67
Figure 27: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values. .........................................................................................................................72
Figure 28: Loss metric for train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the loss values. ........73
Figure 29: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values. .........................................................................................................................74
Figure 30: Loss metric for train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the loss values. ........75
Figure 31: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values. .........................................................................................................................76
Figure 32: Loss metric for train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the loss values.

Figure 33: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values.

Figure 34: Loss metric for train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the loss values.

Figure 35: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values.

Figure 36: Loss metric for train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the loss values.

Figure 37: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values.

Figure 38: Loss metric for train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the loss values.
List of Tables

Table 1: Classification report of BiLSTM – CRF network with word vectors as input. ............71
Table 2: Classification report of BiLSTM – CRF network with word and character vectors as input. .................................................................73
Table 3: Classification report of BiLSTM – CRF network with word, character and POS tag vectors as input. .................................................................75
Table 4: BiLSTM – CRF network with attention in POS embeddings. .................................................................77
Table 5: BiLSTM – CRF network with attention in word character embeddings and attention on POS embeddings.................................................................79
Table 6: BiLSTM – CRF with attention layer. .................................................................81
Table 7: Table of aggregated results for every name entity of the different neural network models implemented.................................................................83
Table 8: Table of aggregated average results for every name entity of the different neural network models implemented.................................................................84
Chapter 1 Introduction

As the volume of published medical literature continues to grow exponentially, there is more and more research for physicians to assess and evaluate and less time to do so. Evidence based medicine (EBM) [1] is a widely accepted paradigm in medical practice that relies on evidence from patient-centered clinical research to make decisions. Taking an evidence-based approach to searching means doing a systematic search of all the available literature, individually critically appraising each research study and then applying the findings in clinical practice. However, this is a time consuming activity. One way to facilitate searching for a precise answer is to formulate a well-focused and structured question [2]. Physicians are educated to formulate their clinical questions according to several well defined aspects in EBM: Patient/Problem (P), Intervention (I), Comparison (C) and Outcome (O), which are called PICO elements. In many documents in medical literature (e.g. MEDLINE), one can find the elements of the PICO structure, but rarely explicitly annotated. To identify documents corresponding to a patient’s state, physicians also construct their queries according to the PICO structure. It is not easy to identify PICO elements in documents, as well as in the question if these are not explicitly separated in it.

A way to identify PICO elements in documents can be achieved through Named Entity Recognition (NER), which is a natural language processing task. NER aims to recognize mentions of specific names from text belonging to named entity types such as patients, intervention etc. It has many applications and acts as a standalone tool for information extraction and filtering. It also plays a key role in various natural language applications such as question answering, machine translation, automatic text summarization and others.

Different approaches are given by different researchers for NER task. Earlier rule based approaches [3], [4], [5] came into existence. These approaches focus on extracting names using a number of handcrafted rules. Rule based approaches provide better results for restricted domains only. Later machine learning approaches [6], [7] i.e. supervised and unsupervised approaches overcome the difficulties of rule based approaches. These approaches are easily trainable and adaptable to different domains. But these techniques require large annotated corpus for training and testing which is hard to find, especially on the medical field. Nowadays, deep learning approaches [8], [9] are widely used. These approaches take the advantage of both rule based and machine learning based techniques.

1.1 Motivation/Object

Purpose of this project is the automated extraction of research questions and answers from medical publications. This task can be achieved with the development of a recognition system of specific entities in medical publications. In order to achieve that, it was necessary to have annotated text, abstracts in this specific case, with the desired entities (patients, intervention and outcomes). Hence, the goal of this project is to build a prediction system that would have medical abstract publications as input and would predict a specific entity for each word.
1.2 Course of investigation

This diploma thesis consists of seven chapters in total, Introduction (present chapter) and six more. The following is a summary of the contents of each chapter:

- In the second chapter there is a description of what Natural Language Processing (NLP) is and its application fields. Among natural language processing task, the one that is really important for this project is Named Entity Recognition (NER). Hence, there is brief description of NER task, its techniques (supervised, semi-supervised and unsupervised) and applications. Also, there is special mention in this chapter regarding deep learning in NER since this field is applicable in the implementation of this thesis.
- The third chapter contains information regarding Evidence Based Medicine (EBM). It describes the basic EBM model and important EBM frameworks such as PICO and SPIDER. Finally it presents a summary of the basic machine learning techniques in EBM found in the literature.
- Chapter four initially presents the theoretical background. It explains basic terms and methodologies used for the preprocessing of the data and the development of the prediction system. Furthermore it presents the dataset that was used for the implementation of this thesis along with all the steps followed in order to design, develop and evaluate the prediction system.
- Chapter five presents the results of the system developed in the context of this work.
- Chapter six contains the conclusions along with a discussion of future improvements of the system.
- Finally, chapter seven contains all the bibliographical sources used in this project.
Chapter 2 Natural Language Processing

Natural language processing (NLP) is the ability of a computer program to process and analyze large amounts of natural language data. It is a subfield of computer science, information engineering, information retrieval and artificial intelligence. This chapter will present a short history of how NLP started, its basic tasks and applications and specific reference will be given to Named Entity Recognition, which constitutes an NLP task with various and very important applications in technology.

Natural language was introduced in 1950s by Alan Turing who published an article titled “Computing Machinery and Intelligence”. This article first introduced the idea of whether machines are able to think and it is now been known and called as the Turing test. Four years later, the Georgetown experiment involved automatic translation of more than 60 Russian sentences to English. This step, considered at that time, as the milestone in the field and the research community was too confident to assume that machine translation would be a solved problem by the end of that decade. Yet, ten years later, progress was much slower than expected. The revolution in natural language processing systems appeared much later, in 1980s with the development of machine learning algorithms. Some specific machine learning algorithms, such as decision trees, where based on if-then rules which replaced the already existing hand-written rules system. During that period, machine learning models that used probabilities became noticeable. This reorientation from rule-based systems to probability-based systems resulted in the birth of statistical NLP. Statistical NLP gives better results in practice because these approaches utilize the most common cases by learning with real data. The more representative the data, the better the NLP systems get.

At the beginning, NLP systems were based more on supervised learning algorithms. In this case, data were hand-annotated with the desired answers and algorithms were trained with these data. Although, due to the tremendous amount of non-annotated data available, research focused on unsupervised and semi-supervised learning algorithms. These algorithms are able to learn from data that are not annotated with the desired answers or learn from a combination of annotated and non-annotated data. Semi-supervised and unsupervised learning is more difficult than supervised learning and typically produces less accurate results for a given amount of input data.

In the 2010s neural network-style machine learning methods became popular in many fields including NLP. These techniques seem to achieve state-of-the-art results in many natural language tasks. Popular techniques try to capture the semantic properties of words, or try to learn higher-level tasks, such as question-answering. This evolution resulted in changes on NLP systems and in the birth of deep neural network-based approaches [10].
2.1 Natural language processing tasks
This section presents some of the most commonly researched tasks in NLP. These tasks are divided in four main categories: syntax tasks, semantics tasks, discourse tasks and speech tasks. Each one of these tasks consists of various subtasks that are being used in different applications.

2.1.1 Syntax tasks
Syntactic tasks are the most evolved tasks in the field of NLP. These tasks are very important because they simplify the process of subsequent components as they try to extract meaning from the input and also they help detect any new or unusual meanings. Without syntactic analysis it would be impossible to understand the meaning of a sentence or a text [10].

The tasks belonging in this category are described below:

- **Grammar induction**
  It is the process of learning a formal grammar from a set of observations, in such a way of constructing a model which accounts for the characteristics of the observed objects. More generally, grammar induction is that field of machine learning where the instance space consists of discrete combinatorial objects such as strings, trees and graphs [11].

- **Lemmatization/ Stemming**
  Lemmatization is the algorithmic process of determining the lemma of a word based on its intended meaning. In lemmatization it is very important to correctly recognize the intended part of the speech and the meaning of a word in a sentence. It is also very important to identify the context surrounding that sentence [12]. Similar to lemmatization, stemming is the process of reducing inflected words to their base or root form (stem). The stem does not have to be identical to the morphological root of the word. It is usually acceptable that related words map the same stem, even if this stem is not itself a valid root [13]. These two processes are commonly treated together as one task from the NLP research community.

- **Morphological segmentation**
  It is the process of separating words into individual morphemes and identifying their respective class. The difficulty of this task depends on the difficulty of the language being considered. For example, English is a language with simple morphology, hence it is possible to ignore this task and simply model all possible forms of a word (e.g. write, writes, writing, written) as separate words. In other languages, such as Indian or Turkish, morphology is much more complex and an approach like that is not possible [14].

- **Part – of – speech tagging (POS tagging)**
  It is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context. A simple form of POS tagging is identification of words as nouns, verbs, adjectives etc. It is not uncommon that in many languages, several words can serve as multiple parts of speech depending on the context in which they are used. For example, "book" can be a noun if used in the following sentence...
"the book on the table", but it also can be a verb if used in the sentence: "to book a flight" [15].

- **Parsing**
  It is the process of conducting grammatical analyses of a given sentence. In natural languages, sentences might have multiple possible analyses, even though some of them might not make sense to a person. Parsing is distinguished in two types. The Dependency parsing, which focuses on the relationship between words in a sentence and the Constituency parsing which focuses on building out the parse tree with the use of a Probabilistic Context – Free Grammar (PCFG) [16].

- **Sentence breaking**
  It is the process of locating the sentence boundaries in a text. Sentence boundaries are usually marked by punctuation marks, such as periods. However, sentence boundary identification can be challenging since several punctuation marks are often used for other purposes apart from defining the end of a sentence. For example, periods can also be used in abbreviations, while question marks or exclamation marks can also be used in slang language [17].

- **Word segmentation**
  It is the process of separating a text into its component words. In a language where word boundaries are marked easily (such as English, where word boundaries are defined by space) this process is quite easy. In languages though such as Chinese, where word boundaries are not marked as easily, word segmentation can be a challenging task which requires further knowledge of the language, like vocabulary knowledge and insight of the morphology of the words [18].

### 2.1.2 Semantics tasks

Semantic tasks are trying to define the meaning of the sentence or text which might not be unique. In other words, the output of the semantic component is the “meaning” of the input text. Although there is a great challenge in exactly defining the term “meaning”, since it cannot simply be expressed by a sequence of words. That is the reason why there are three general kinds of semantic representations in use: propositional logic, first-order predicate logic (FOPL) and various representations that can handle expressions not representable in FOPL [10].

The tasks belonging in this category are described below:

- **Lexical semantics**
  This task answers the question of how the meaning of the lexical units correlate with the structure of the language or syntax. In lexical semantics, the units of analysis are lexical units which include, not only words but also sub – words or sub – units, such as affixes, and also aggregate words and phrases. The lexicon, the catalogue of words in a language, consists of lexical units.
The area of lexical semantics looks at the classification and decomposition of lexical items, the differences and similarities in lexical semantic structure and finally the relationship of lexical meaning to sentence meaning and syntax [19].

- **Machine translation (MT)**
  It is the process which automatically translates text from one human language to another. MT performs simple replacement of words in one language for words in another. The drawback regarding this procedure is that usually it cannot produce a valid translation because recognition of whole phrases and their closest counterparts in the target language is needed. Current MT software allows customization by domain or profession, improving output by limiting the scope of allowable substitutions [20].

- **Named entity recognition (NER)**
  This process attempts to locate which items inside a text correspond to proper names, such as people or places and also the type of these names, if the names characterize locations, organizations or persons. Most NER systems are taking as input an annotated block of text and produce as output an annotated block of text that highlights the names of entities. In languages such as English, capitalization can indicate named entities, but this information is not useful in all languages. In German for example, all nouns are capitalized regardless of whether they are names or not. Furthermore, the first letter of the sentences is also capitalized, and named entities often span several words, only some of which are capitalized. In addition, there are certain languages, such as Arabic or Chinese, which do not have capitalization at all [21].

- **Natural language generation**
  During this process structured data are transformed into natural language. It can be used to produce long format content for organizations to automate custom reports, as well as to produce custom content for a web or mobile application [22].

- **Natural language understanding**
  This process deals with machine reading comprehension. For this sake, it converts text into structures that are easier for computer programs to manipulate. This task involves the identification of the intended semantic from the multiple possible semantics which can be derived from a natural language expression which usually takes the form of organized notations of natural language concepts.

- **Optical character recognition (OCR)**
  This process involves the electronic conversion of images of typed, handwritten or printed text into machine – encoded text, whether it is originated from a photo of a document, a scanned document or a scene photo. It is a common method of digitizing printed texts so that they can be electronically edited, searched, stored, displayed on-line, and used in other machine processes such as cognitive computing, MT, text mining etc. [23].
• **Question answering**
The purpose of this task is to create systems that automatically answer questions posed by humans in natural language by searching through a collection of text documents. The answers may be constructed by querying a structured database of knowledge or information [24], [25].

• **Relationship extraction**
This task requires the detection and classification of semantic relationship mentions within a set of artifacts, typically from text or xml documents. In other words, it identifies the relationships among named entities within a text.

• **Sentiment analysis**
This process is also known as opinion mining. This task uses NLP and text analyses to identify and quantify affective states. Sentiment analyses is usually used for identifying trends in public opinion, especially in social media, and in product reviews or surveys. Aspect – based sentiment analysis (ABSA) is a subfield of sentiment analyses that has become popular due to the fact that it permits aspects of a product or a service to be examined in more detail. ABSA was introduced for the first time at SemEval 2014. In ABSA, datasets are annotated in a sentence level with aspect terms (e.g. “pizza”, “mouse”) and their polarity for a specific domain (e.g. restaurant, laptop) as well as coarser categories (e.g. “food”) and their polarity for, e.g. restaurants only. All the identified constituents of the expressed opinions (e.g., aspects, opinion target expressions and sentiment polarities) meet a set of guidelines and are linked to each other within sentence-level tuples. These tuples are important since they indicate the part of text within which a specific opinion is expressed. However, a user might also be interested in the overall rating of the text towards a particular aspect. Such ratings can be used to estimate the mean sentiment per aspect from multiple reviews [26], [27].

• **Topic segmentation and recognition**
This process divides a text into separate units, such as words, sentences or topics and identifies the topic that each segment belongs to.

• **Word sense disambiguation**
In a lot of languages, many words have more than one meaning. In English for example, you can “read a book”, or “book a flight” or you can talk about apple, the fruit, or the company. This is why it is important to select the meaning which makes the most sense in the context of the text being examined. This is the purpose of this task, to identify which sense of the word is being used in the sentence. To tackle this issue, usually it is given a list of words and associated word senses, e.g. from a dictionary or from an online resource such as WordNet [28].
2.1.3 Discourse tasks
The discourse tasks try to determine the referents of pronouns in a text and define the noun phrases. Apart from that, these tasks attempt to recognize elliptical sentence fragments, misspellings and any other form of nonstandard language [10].

The tasks belonging in this category are described below:

- **Automatic summarization**
  This process shortens a text document, in order to create a summary that contains the most important points of the original document. This task is usually used to provide summaries of text of a known type, such as research papers or articles of newspapers.

- **Coreference resolution**
  This task identifies, in a sentence or a text, which words refer to the same objects. It recognizes which words have the same referent, e.g. refer to the same person or thing.

- **Discourse analysis**
  This task consists of two sub-tasks, identifying the discourse structure of the text and recognizing and classifying how speech acts in a text. The first task is related to the nature of the discourse relationships between sentences (e.g. elaboration, explanation, contrast) and the second task is related to the yes-no question, the content question, statement, assertion, etc.).

2.1.4 Speech tasks
The speech tasks try to identify and distinguish the words in a human’s speech or in a sound clip and, depending on the case, translate it into text.

The tasks belonging in this category are described below:

- **Speech recognition**
  It is the process which identifies the textual representation of the speech, by analyzing a sound clip of a person speaking. This is a very difficult task, concerning the fact that in oral speech there are no distinct pauses between words and thus, speech segmentation is a necessary subtask of speech recognition. In most spoken languages, the sounds representing the letters blend into each other making difficult the conversion of the sounds to discrete characters. Another significant obstacle in speech recognition is the fact that the same words in the same language are pronounced different by the different people that speak this language, so the speech recognition software has to be able to recognize the wide variety of inputs as being identical to each other in terms of its textual equivalent.

- **Speech segmentation**
  As already mentioned, this task separates a sound clip of a person speaking into separate words.
2.2 Named Entity Recognition

This section introduces Named Entity Recognition (NER) problem. NER is one of the major tasks in NLP and has been an open research problem for many years. Apart from describing what NER is, this section will also present the different NER tasks and approaches.

Before defining NER, it is important to answer the question of what is a named entity. A named entity is a word (or a phrase) that clearly distinguishes one item from a collection of other items that have similar characteristics. Usually, the word named restricts the range of entities that have one or more rigid identifiers representing an analyzer. Rigid identifiers include the proper names, but it depends on the field of interest that can refer to the field word reference as entities called. For example, in bio-informatics entities are genes and gene products.

Named Entity Recognition (NER) is the process of locating a word or a phrase that references a particular entity within a text. First appearance of NER task was in the Sixth Message Understanding Conference (MUC - 6) and involved the recognition of entity names (people, organizations), place names, temporal expressions and numerical expressions [30].

2.2.1 Named Entity Recognition techniques

This section is going to present some of the basic approaches to NER. These approaches belong to three main categories: supervised, semi – supervised and unsupervised. Finally, deep learning techniques to NER problem solving are being presented since deep learning is a new field that has successful application to NER field.

**Supervised methods**

Supervised methods are using algorithms that learn a model by using annotated training examples. They can either learn disambiguation rules based on discrete features or try to learn the parameter of the assumed distribution that maximizes the likelihood of data compilation. The most frequent used supervised methods for NER are Hidden Markov Model (HMM), Decision Trees, Maximum Entropy Models (ME), Support Vector Machines (SVM) and Conditional Random Fields (CRFs). Each one of these methods will be presented in this section.

**Hidden Markov Models (HMM):**

This model is one of the earliest models applied in NER. HMM was used for the first time for solving NER problems by Bikel et al. [31] by introducing a system called Identifiers to detect NER. In this system, in every word in the context, only a single label can be assigned. Due to this restriction, the model assigns to every word in the corpus either one of the desired classes, or the label NOT – A – NAME to declare that this word doesn’t belong to “any of the desired classes”. 

---

- **Text – to – speech**
  This task is the opposite task of speech recognition. In speech recognition, a sound clip of a person speaking was transformed to text. In text – to – speech, a text is transformed into a spoken representation [29].
In order for a sentence to be tagged, the most likely sequence of name – classes (NC), given a sequence of words had to be found:

$$\text{max} \Pr(\text{NC}|W)$$

(1)

HMM is a model that tries to generate the data, sequences of words $W$ and labels $\text{NC}$ from distribution parameters.

$$\Pr(\text{NC}|W) = \frac{\Pr(W, \text{NC})}{\Pr(W)}$$

(2)

In order to maximize $\Pr(W, \text{NC})$, the Viterbi algorithm Forney is used, through the entire space of all the possible name – class assignments. In their study, they modeled the generation in three steps:

1. Select a name class $\text{NC}$, based on the previous name – class and previous word.
2. Generate the first word inside the name class, conditioning on the current and previous name – classes.

$$\Pr(nc|nc_{-1}, w_{-1}) \cdot \Pr(< w, f >_{\text{first}} | nc, nc_{-1})$$

(3)

3. Generate all consecutive words inside the current name – class, where each word is conditioned on its immediate predecessor.

$$\Pr(< w, f > | < w, f >_{-1}, nc)$$

(5)

There is also a distinct end marker ”+end+”, so that the probability may be computed for any current word to be final word of its name-class:

$$\Pr(< +\text{end}+, \text{other} > | < w, f >_{\text{final}}, nc)$$

(6)

Maximum Entropy model:

While HMM are generative models, maximum entropy models (ME) are discriminative models. This means that, given a set of training data and features, the model directly learns the weight for discriminative features for classification. In ME models, the goal is to maximize the entropy of the data, so as to generalize as much as possible for the training data. In ME models, each feature is associated with a parameter $\lambda_l$. The Conditional probability can be calculated by the following equations:

$$P(f | h) = \frac{\prod_l \lambda_l^{g_l(h,f)}}{Z_\lambda(h)}$$

(7)

$$Z_\lambda(h) = \sum_f \prod_l \lambda_l^{g_l(h,f)}$$

(8)

Maximization of entropy ensures that for every feature $g_l$, the expected value, according to a ME model, will be equal to empirical expectation of $g_l$ in the training corpus.
SVM based models:

Support Vector Machine (SVM) models are very popular machine learning models with various applications. The basic idea of these models is based on learning a linear hyperplane that separates the positive examples from the negative examples based on a large margin. Large margin indicates that the distance between the hyperplane and the point from either distance is maximum. The points that are closest to this hyperplane, on either side, are called support vectors. The geometric interpretation of an SVM model is shown in Figure 1. The linear classifier is based on two parameters, a weight vector $W$ and a bias $b$. The weight vector has a right angle with the hyperplane that separates the instances, while bias determines the offset of the hyperplane from the origin. A sample is determined that belongs to the positive class if $f(x) = wx + b > 0$ and is classified in negative class otherwise. In case there are more than two classes, a group of classifiers are used to classify the data.

![Figure 1: Geometric interpretation of SVM][30]

In NER, SVM can be used to deal the problem as a binary decision problem. McNamee and Mayfield [32] used the binary approach and checked whether the word belonged to one of the eight classes, e.g. B- Beginning, I- Inside tag for person, organization, location and misc tags. Since there were eight classes, eight classifiers were trained. The selected features that were inserted into the classifiers were all binary and consisted of orthography features, punctuation features and language-related features. In order to produce a single label for each token, the set $S$ of possible tags was generated and identified. If $S$ was empty, tag 0 was assigned, otherwise, the most frequent tag was assigned. If both beginning and inside tags were present then beginning tag was chosen.
Conditional Random Field based models:

Conditional Random Field (CRF) was firstly introduced by Lafferty et al. [33] as a statistical model for pattern recognition and machine learning using structured prediction. Later, McCallum and Li [34] proposed a feature induction method for CRF in named entity.

Let assume an observed input data sequence $o = < o_1, o_2, ..., o_T >$. This sequence can be a sequence of words in a text. Let $S$ be a set of states, where each state is associated with a label, and $s = < s_1, s_2, ..., s_T >$ be a sequence of states. CRF define the conditional probability of a state sequence given an input sequence to be

$$P(s|o) = \frac{1}{Z} \exp \left( \sum_{t=1}^{T} \lambda_k f_k(s_{t-1}, s_t, o, t) \right)$$  \hspace{1cm} (9)

Where $Z$ is the normalization factor, obtained by marginalizing over all state sequences, $f_k(s_{t-1}, s_t, o, t)$ is a feature function and $\lambda_k$ is the learnt weight for each feature function. With the use of dynamic programming, state transition between two CRF states can be efficiently calculated.

The modified forward values, $a_T(s_i)$, to be the “unnormalised probability” of arriving state $s_i$, given the observations $< o_1, o_2, ..., o_T >$. State $a_0(s)$ defines the probability of starting in each state $s$, and recursively calculated as:

$$a_{t+1}(s) = \sum_{s'} a_t(s') \exp(\lambda_k f_k(s', s, o, t))$$  \hspace{1cm} (10)

Semi–supervised methods

Semi–supervised learning algorithms use both labeled and unlabeled data in order to train their models and create their own hypothesis. Algorithms usually start with a small amount of seed data and create more hypothesis by using large amount of unlabeled data. This section presents one of the most frequent semi–supervised NER system, which is using the bootstrapping method.

Bootstrapping based model:

The main goal of semi–supervised algorithms is to deal with the lack of annotated data and the data sparsity issue. Due to these problems, semi–supervised methods usually start with a small amount of annotated data, a large amount of unannotated data and a small set of initial hypotheses or classifiers. With each iteration, more annotations are generated and stored until a certain threshold occurs to stop the iterations.

Usually, in bootstrapping based models, AdaBoost is most frequently used. Carreras et al. [35] have modeled the Named Entity Identification task as a sequence labeling problem through BIO labeling scheme. In BIO labeling scheme, input is considered as word sequence to label with one of the Beginning of NE (B-) tag, Inside of tag (I-) and outside of NE (O-) tag. Three binary classifiers are used for tagging, one corresponding to each tag. Orthographic and semantic feature extraction allowed a relational representation of examples. Binary AdaBoost was used as a learning algorithm for the classifiers. The boosting algorithm combines several fixed - depth
decision trees. Each tree is learned sequentially by presenting the decision tree a weighting over the examples which depend on the previous learned trees.

Unsupervised methods

Supervised methods require a large number of features and annotated data in order to train a good model. Unfortunately, in many NER applications and languages it is difficult to find large annotated datasets. To deal with lack of annotated text across domains and languages, unsupervised techniques for NER have been proposed, such as the KNOWITALL system.

KNOWITALL:

This system was proposed by Etzioni et al. [36]. KNOWITALL extracts information from the web in an unsupervised, open-ended manner. KNOWITALL uses eight domain independent extraction patterns to generate candidate facts. As an example, the generic pattern "NP1 such as NPLIST2" indicates that the head of each simple noun the phrase (NP) in the NPLIST2 list is a member of a class called NP1. It then automatically checks the credibility of the candidate events it extracts using reciprocal information points (PMIs) calculated using large body text as the backbone. Based on the PMI score, KNOWITALL associates a probability with all the events it exports, allowing it to manage the trade-off between accuracy and recall. It is based on startup technique that generates seeds from general export patterns and automatically discriminating phrases.

Unsupervised NER across languages:

Munro and Manning [37] proposed an unsupervised system that generates seed candidates through local, cross-language editing and then bootstraps to make broad predictions in two languages, optimizing contextual models, word format, and alignment. It is completely uncontrollable, without manual labeling, without external resources, using only parallel text that does not need to be easily adaptable. Results of this model are promising, especially if it is combined with supervised methods.

2.2.2 Features in Named Entity Recognition systems

Feature engineering is a very important field in NER. Features are descriptions or characteristic properties of words designed for algorithmic consumption. The features can be defined in many ways using boolean values, numeric or nominal values. A NER system can be presented by using three attributes:

1. A boolean attribute, by assigning the value True if the first character of the word is capitalized and False if it isn’t.
2. A numeric attribute, which corresponds to the length, in characters, of the word.
3. A nominal attribute, which corresponds to the lowercased version of the word.

By following the above set of features, the sentence “The president of Apple eats an apple.” Is represented with the following feature vectors:
<true, 3, "the">, <false, 9, "president">, <false, 2, "of">, <true, 5, "apple">, <false, 4, "eats">,
Usually, the NER problem is resolved by applying a rule system over the features. This rule system can be consisted of recognition rules, e.g. “capitalized words are candidate entities” and classification rules, e.g. “the type of candidate entities of length greater than three words is organization”. Although, real systems tend to be much more complex and their rules are often created or expanded by automatic learning algorithms.

This section will describe the features that are most often used for the identification of named entities. These features are organized in two categories: word – level features and list lookup features [30].

**Word level features**

Word level features are related to the character level feature of the words. Word level features can be digit patterns. Certain patterns of digits give strong indication about the type of named entities. For example, digits that are followed by units stand for quantity such as 10Kg. These features can also be expressed by common word endings. Named entities can have common suffixes features. For example, professions often end in “ist”, like journalist. Finally, word level features can also be expressed by pattern features. A pattern feature is created when words are mapped into small set patterns over character types. For example, a pattern feature can map all uppercase letters to “U”, all lowercase letters to “l”, all digits to “0” and punctuation marks to “-“.

Following this pattern, x = “I.B.M” will be y = “U-U-U-”.

**List lookup features**

Apart from word level features, in NER there are also the list lookup features. The list function means that “a” is a relation for the entities in the list (e.g. Athens is a city). If a word is included in the list, the probability of that word appearing in a sentence called an entity is high. In this kind of features, the terms “gazetteers”, “lexicon” and “dictionary” are used conversely with the term “list”. Common nouns listed in a dictionary are useful, for instance, in the disambiguation of capitalized words in ambiguous position (e.g. sentence beginning). The recognition of organization names using the words that commonly occur in their names is also a proposition.

For the list lookup to work, the word in question should exactly match at least one element of a pre-existing list. Three lookup strategies could be used to effective use list lookup for NER.

1. Different form of words should be considered as valid matches. For example, apple should match apples.
2. Candidate words can be "unclear" with reference to the list words using some processing distance threshold. This helps to minimize the small spelling variation between two words.
3. A reference list that can be accessed by using the Soundex algorithm which normalizes candidate words to their respective Soundex codes. The Soundex code is supposed to be a combination of the first letter of a word, plus a three digit code that represents its phonetic sound. As a result, similar sounding names are equivalent in respect.
2.2.3 Named Entity Recognition with Deep learning techniques

All the NER approaches discussed so far are related to machine learning algorithms and most of these machine learning algorithms work well because of human designed representations and features. Deep learning is a technique which allows to learn features independently than actually hand-crafting features for the algorithm. Deep learning is about representation learning with good features automatically and the results are very promising and considered nowadays as state-of-the-art. The most important asset of deep learning is the capability of representation learning along with the semantic composition empowered by both the vector representation and neural processing.

Deep learning is a set of machine learning algorithms that try to learn multilevel input models, commonly known as neural networks. Each layer tries to learn a concept from its previous input layer. With each subsequent layer, deep learning algorithm attempts to learn multiple levels of concept of increasing complexity/abstraction.

Neural networks are inspired from the central nervous system and the biological neurons of humans. The primary computation node is called a neuron. It is a cell composed of a cell body containing the nucleus and most of the cell’s complex components along with many branching extensions called dendrites, plus one very long extension called axon. Neurons are connected to one another and communicating through the synapses. Synapses are connected to the dendrites, or directly to the cell body of other neurons. Neurons receive short electric impulses (signals) from other neurons through the synapses. When a neuron receives a sufficient number of signals from other neurons within a few milliseconds, it fires its own signals.

![Biological neuron](image)

*Figure 2: Biological neuron*

The artificial neuron is a very simple representation model of the biological neuron. It has one or more binary (on/off) inputs and one binary output. The artificial neuron activates its output when more than a certain number of its inputs are active. Even with this simplified artificial neuron it is possible to build a network of various artificial neurons that compute any logical computation we want.

Artificial neural networks (ANNs) are mathematical modeling of biological neural networks. In ANNs, each neuron is a perceptron with weighted inputs. Weights are analogous to connection strength in biological neural networks.
The Perceptron is one of the simplest ANN architectures. It is based on an artificial neuron called Threshold Logic Unit (TLU). In TLU, inputs and outputs are numbers and each input connection is associated with a weight. TLU computes a weighted sum of its inputs, $z = w_1x_1 + w_2x_2 + \cdots + w_nx_n = w^Tx$, then applies a step function to that sum and outputs the result: $h_w(x) = \text{step}(z) = \text{step}(w^Tx)$.

Figure 3: Threshold Logic Unit [38].

The most common step function in Perceptron is the heaviside step function:

$$heaviside(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z > 0 \end{cases}$$ (11)

Sometimes, the sign function can be used instead:

$$\text{sign}(z) = \begin{cases} -1, & \text{if } z < 0 \\ 0, & \text{if } z = 0 \\ 1, & \text{if } z > 0 \end{cases}$$ (12)

A single TLU can be used for simple linear binary classification. It computes a linear combination of the inputs and if the result exceeds a threshold, it outputs the positive class or else outputs the negative class.

A Perceptron is simply composed of a single layer of TLUs, with each neuron connected to all the inputs. These connections are often represented using special pass-through neurons called input neurons: they just output whatever input they are fed. Usually, an extra bias feature is generally added ($x_0 = 1$). This bias feature is represented using a special type of neuron called bias neuron, which outputs 1 all the time [38].

A multi-layer Perceptron (MLP) is composed of one input layer, one or more layers of TLUs, called hidden layers, and one final layer of TLUs called the output layer. Every layer, except the output layer, includes a bias neuron and is fully connected to the next layer. When an ANN has two or more hidden layers, it is called a deep neural network (DNN). An example of a simple ANN is shown in Figure 4. Input is fed at the bottom layer which constitutes the first layer. For more
efficacy, layered architecture is used for connections. With each layer, the network tries to learn the input pattern that helps to correctly classify new examples.

![Artificial neural network](image)

**Figure 4: Artificial neural network [30].**

In order to train MLPs, the *backpropagation* training algorithm is used. Backpropagation algorithm works like this: the algorithm feeds to the network each training instance and every neuron computes an output and forwards it to the next layer (this is called forward pass). This procedure happens for all the hidden layers until the output layer. Once it reaches the last layer, the network’s output error is calculated. The network’s output error is defined as the difference between the desired output and the actual output of the network. Through the output error, the algorithm computes how much each neuron of the last hidden layer contributed to each output neuron’s error. Then, it computes how much of these error contributions came from each neuron in the previous hidden layer and this process continues until the algorithm reaches the input layer. This reverse procedure efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward in the network. The last step of the algorithm is the Gradient Descent step on all the connection weights in the network, using the error gradients measured earlier [38].

Apart from backpropagation algorithms, there are others that can learn better features but are trained greedily for each layer. These layers are then stacked one above the other to learn higher level features. Each layer learns a feature and then the next layer learns high level feature from the layer below it. An autoencoder neural network is an algorithm that applies backpropagation target values to be equal with input values.
It takes an input $x \in [0,1]^d$ and maps it into latent representation $y \in [0,1]^{d'}$ through a deterministic mapping function:

$$y = f(Wx + b)$$ (13)

where $f$ is a non–linear function like sigmoid or tanh. Representation $y$ is then mapped to an output $z$ which is a reconstruction of the original input $x$ using a similar transformation

$$z = f(W'y + b')$$ (14)

The model needs to learn the parameters $W, b, W', b'$. Formulation is optimized in such a way that the average reconstruction error is minimized. Reconstruction error is defined as:

$$L(x,z) = H(B_x \parallel B_y) = -\sum_{k=1}^{d} [x_k \log z_k + (1 - x_k) \log(1 - z_k)]$$ (15)

By minimizing the above equation, we minimize the number of bits needed to represent information in $y$ instead of representing it from $x$. As a result, we capture the main factors of variation in the data similar to projection on principal component. Since the factors learned are smaller than the input factors, there is a loss in information about the input. Autoencoder provides a reconstruction mechanism with low reconstruction error rate to test examples drawn from a similar distribution as training but with a high reconstruction error for a uniformly chosen input vector [30].

There are three strong reasons of why someone should apply deep learning techniques to NER, in contrast with traditional machine learning techniques. First, non–linear transformation applied in deep learning generates non–linear mappings from input to output and provides the ability to deep learning models to learn complex features from data via non–linear activation functions, in contrast with linear models that don’t have this flexibility. Second, the traditional approach
requires feature extraction techniques which is time and energy consuming and requires engineering skills and domain expertise and is not always guaranteed that the extracted features are best suitable for the problem in question each time. Deep learning models are able to automatically learn useful representations and extract the best features from the raw data. Finally, deep neural NER models can be trained in an end-to-end paradigm, by gradient descent. This property enables researchers to design possibly complex NER systems.

More information regarding specific artificial neural networks that are being implemented in this thesis, are described in Chapter 4.

2.2.4 Named Entity Recognition applications
This section presents several applications where NER task is successfully used. Various natural language applications consider NER as an important part of their structure. In these applications, NER is employed in the preprocessing stage and extracts proper nouns that are required by these applications in order to improve their performance. Many of the NER applications, have already been discussed in NLP applications as well. This is to be expected since NER is a very important application on NLP that can be used as a method to other NLP tasks as well. Below are presented the different NER tasks.

- **Information extraction systems:**
  Information extraction is the task of extracting relevant information as per the request made by the user. Named entities usually carry important information about the text so employing NER systems significantly improves the information extraction accuracy. A number of extraction systems, starts with recognizing the named entities [39].

- **Question – Answering systems:**
  Question – answering systems aim to find the exact answers to natural language questions in a large collection of documents. Answers of fact based questions are named entities so incorporating named entity system improves the speed and accuracy of getting correct answers [40].

- **Machine translation systems:**
  As already mentioned, machine translation is the process in which text from one language is converted to another. Different translation rules are applied on named entities and other words so extraction of named entities beforehand makes the task of translation systems quite easy [41].

- **Automatic Summarization Systems:**
  Automatic summarization systems produce a short representation of the text while preserving the information content. Named entities are important text information and increase the recognition efficiency of text segments that are further included in summary data [42].

- **Semantic annotation:**
  Semantic annotation enriches unstructured or semi-structured data with an environment that is further linked to the structured knowledge of a domain. It is related to the formal recognition of concepts and relationships. This recognition is implemented with information extraction techniques. Among these techniques NER is used to identify concepts to annotate [43].
2.2.5 Challenges in Named Entity Recognition

Named Entity Recognition is an NLP task with wide variety in various other NLP applications but also as an NLP task itself. Although, it faces a lot of challenges that need to be resolved. Apart from the technique to be used, NER is affected by numerous other factors, such as the language factor, textual genres or domain factors, entity type factors etc.

The most important challenge, is the fact that most of the NER research already developed, has been done in English and other European languages. These languages provide capitalization in order to identify the named entities, which is not valid for Asian and other Indian or Arabic languages. Textual genres or domain factors also affect the accuracy of a NER model. NER systems developed for one domain are not easy to apply and perform the same to another domain.

Regarding the NER models, supervised models require a large annotated corpus for classification of named entities out of testing data. The challenge lies in the fact that annotation of large training data requires a lot of time and domain experts to perform the annotation. However, as already mentioned, this issue can be resolved with the use of semi–supervised and unsupervised methods.

Another challenge for NER systems is ambiguity. Some words have different meaning depending on the context. For example, the word “Paris” can be considered as a name in one context and as a location in another context. In order to get the highest performance of NER system, this issue must be resolved.
Chapter 3 Evidence Based Medicine

Medical knowledge grows every day, and due to this progress, previously accepted facts and treatments rapidly become old and outdated. New drugs are discovered every day and new technological achievements offer the possibility to gain knowledge that was not available before. With all this progress and information, it is really difficult to follow up all these changes and distinguish the best practices for the patient.

Difficulties are real when it comes to physicians and the number of new achievements published in medical papers that they have to follow. Doctors should spend hours in order to read the new published papers, but also they need to spend more time in order to process and evaluate the new techniques and decide if they are suitable for their patients. Due to this problem, a need occurred, that the doctor, with his limited free time should read only a selection of those papers, which are related to his expertise. This is the part where Evidence Based Medicine (EBM) was developed [44].

Evidence Based Medicine is the process of lifelong, self-directed learning in which caring for the patients creates the need for important information about clinical and other health care issues. Applications of EBM correlate individual clinical experiences with the best scientific evidences obtained from clinical research. As already mentioned, the best method of treatment today may change tomorrow. The task of staying current is much easier by incorporating EBM knowledge [45].

The reason that medical community started using EBM was to eliminate the practices that are ineffective and harmful and to promote the effective practices in order to improve quality of medical care. EBM promotes critical thinking and expects from medical community to be open minded in new treatments, drugs and techniques that are proven to be more effective and discard traditional methods that might be harmful. In order for EBM to be developed and exploited at full potential physicians must acquire certain EBM skills, such as the ability to find, critically evaluate, and incorporate sound scientific evidence into their practice.

3.1 The five step Evidence Based Medicine model

In order for physicians to practice EBM, they need to follow five essential steps which constitute the five step EBM model. These steps are:

- Step 1: Problem definition
- Step 2: Find the best evidence with which to solve the problem
- Step 3: Critical evaluation of information
- Step 4: Apply the information into clinical practice
- Step 5: Evaluate the performance.

These steps are going to be described in detail below.
Step 1: Defining the problem

The most important and difficult step in practicing EBM is step one. In this step, the problem must be defined and translated into an answerable question. When it comes to a patient, physicians make a lot of questions in order to understand and define the problem and propose the treatment. In the beginning of that process, questions that arise are more general, complex, unstructured and fuzzy in the physicians head. They need to be converted into specific, simple, clear and directly focused at the problem in order to be answerable by searching on medical literature. In order to do so, EBM should start with a well formulated clinical question.

Step 2: Finding the evidence

After defining the problem which results in the formulation of the clinical question the next step is to search for relevant evidence that will prove the answer to the clinical question. This task is not easy, especially if the problem and question from step one are poorly defined. However, there are several sources of information that may be of help. One source of information can be the traditional sources, such as textbooks and journals. Although this solution is not suggested since these sources of information are usually disorganized and out of date. Another possible source could be the opinions and suggestions of other colleagues or experts, but this solution is not strongly suggested since the quality of information obtained is variable. Other sources of reliable summarized evidence are the online electronic bibliographic databases. These databases offer the ability to search thousands of articles in a relatively short period of time in an increasing number of journals. The skill to effectively search these databases is an important aspect of EBM. Productive searches aim to maximize the potential of retrieving relevant articles in the shortest possible time [44], [45].

Step 3: Evaluate the evidence

After finding the relevant evidence that answer a clinical question, the next step is to appraise the evidence for its validity and clinical usefulness, because putting unreliable evidence into practice could lead to harm being caused or limited sources being wasted. Research and evaluation of evidence may be assessed with regard to three main areas: validity, importance and applicability to the patients of interest. Critical evaluation provides a structured but simple method for assessing research evidence in all three areas [45].

Step 4: Applying the evidence

The forth step in the process of the use of EBM is the decision of how to apply the acquired evidence, after evaluating it and deciding that it is valid. The question that arises is if this evidence, although it is accurate, can be applied to the patient (or population) in question. In order to decide this, patient’s own personal values and circumstances need to be taken into account. The evidence, regarding both its efficacy and risks, should be fully discussed with patient. This is a fundamental principle of EBM: the integration of good evidence with clinical expertise and patients values [45].
Step 5: Performance evaluation

The last step is the evaluation of the specific evidence taking into consideration its efficiency in a specific patient (or population). This evaluation is important in order to decide whether any of the four steps discussed above need further improvement.

By analyzing the five step model of EBM one can conclude that EBM is a conscious, reasonable use of the best evidence in order to make decisions about treatment of patients. By the use of the best possible evidence, physician actually chooses the best possible solution for his patient. In order to practice EBM, physician needs to acquire some specific skills, such as the ability to search medical literature and basic skills in the interpretation of epidemiological and statistical results. We can conclude that the proper use of EBM saves doctors time and raises his level as well as the quality of provided medical services, and increases satisfaction of the health professionals.

3.2 Evidence Based Medicine models

Section 3.1 described in detail the five necessary steps for a successful EBM application. The first step, defining the problem, appears to be the most important one in order to practice EBM. Physicians have numerous questions for every patient they treat. These questions help them define and understand the problem. Unfortunately, the majority of those questions remain unanswered. However, with systematic analyses, improved query formulation and better search these questions could be answered. Hence, one of the main goals of EBM is to help physicians construct precise and well-build clinical questions. In order to achieve that, several frameworks have been developed. This section presents the most popular ones.

3.2.1 PICO framework

Sacket et al. [1] developed a useful framework to make clinical questions more focused and relevant. They suggest that an answerable clinical question should have four essential components:

i) the patient or problem in question,
ii) the intervention of interest,
iii) a comparison of different interventions,
iv) the outcome of interest.

Hence, the answerable clinical question should be structured in the PICO (Problem/Patient, Intervention, Comparison, Outcome) format. Below there is an example that describes the PICO concept:

Let’s assume that the patient is a four month old baby that suffers from viral bronchiolitis. The child’s symptoms gets worse over time and the doctor wonders if he should subscribe corticosteroids that might help the child and reduce the length of stay into the hospital. Finally, he decides to use “clinical score” as a measure of improvement. In order to construct the clinical question, the key components would be:

Patient/Problem: 4 month old baby with viral bronchiolitis
Intervention: corticosteroids
Comparison: no corticosteroids
Outcomes: clinical score, length of hospital stay
Finally the question would be:
“In a 4 month old baby with viral bronchiolitis, does the administration of corticosteroids compared with not giving corticosteroids improve clinical score and reduce length of hospital stay?” [45].

PICO framework is commonly used to identify components of clinical evidence for systematic reviews in EBM and is endorsed by the Cochrane Collaboration [46]. Although it is one of the most popular and widely used frameworks in EBM, PICO is utilizing search terms such as “intervention” that are not relevant with qualitative research. Qualitative research does not use terms such as “intervention” and “control groups” and therefore PICO framework is not suitable for appropriately locating qualitative research.

This is the reason why it has been modified, in practice, to PICOS, where the S refers to the Study design, therefore limiting the number of the irrelevant articles [47]. Apart from PICOS, the PICO framework can be expanded to PICOTT. This framework provides the same information as PICO, patient-intervention-comparison-outcome but also includes information regarding the Type of question being asked (therapy, diagnosis, prognosis etc.) and the best Type of study design for that specific question. Using PICOTT helps the clinician distinguish the important parts of the clinical question that are most applicable to the patient and facilitates the searching process by identifying the key concepts for an effective search strategy [2].

**Challenges in applying the PICO framework**

Huang et al. [48] revealed a number of challenges while applying the PICO framework which are going to be described in this section.

- **Inability to reconstruct the original question:**
  Given a PICO framework we usually cannot recover the original clinical question. If only the problem is known, it is a difficult task to guess the clinical question. This issue however, is easily resolved if the clinical task (e.g. therapy) is known.

- **Inability to encode fine-grained relationship between frame elements:**
  It is difficult to represent a question with fine-grained semantic relationships between concepts in a PICO framework. The PICO representation mainly relies on the semantic relationships between concepts to connect different elements.

- **No explicit/temporal state model:**
  The PICO frame describes a precise moment in time. However, many clinical questions and especially answers are evolving during time, or in fact, a clinical question can integrate the time concept inside the question, e.g. “Are there any advances in the treatment of motion sickness since 90s?” Unfortunately, the PICO framework does not contain provisions in order to capture temporal changes.

- **Overloaded slots:**
  In the PICO framework, P stands for patient/problem. The problem regarding this representation appears in questions where in their pattern exists both the patient and the
problem, like in the diagnosis questions. In these questions, $P$ has to be more clearly defined, explicitly separating the problem from population.

- **Inability to capture anatomical relationships:**
  The PICO framework is unable to capture anatomical relations that may be relevant to a clinical question. There isn’t an element in PICO representation that can capture “body parts”.

### 3.2.2 Other Evidence Based Medicine Frameworks

As already stated, the PICO framework cannot support qualitative research. Thus, a new tool was introduced in order to make the PICO components suitable for qualitative research, together with the addition of a new component. This tool was called, the SPIDER tool.

In qualitative research, findings are not always intended to be generalized beyond the study population. Thus, “Sample” was preferable instead of “Problem/Patient”. In the new tool, component “Intervention” was replaced by “Phenomenon of Interest”, since in qualitative research the aim is to understand certain behaviors, decisions and individual experiences. Smaller sample sizes and the exploratory nature of qualitative research resulted in the replacement of component “Comparison” with “Design”. Qualitative research outcomes might be unobservable and subjective constructs, so the term “Evaluation” was deemed more suitable than “Outcomes” used in PICO. Finally, the SPIDER tool has the added advantage that it might be suitable for mixed-methods and quantitative research search strategies, made possible by the addition of “Research type”.

Apart from SPIDER tool, several other tools have been proposed for qualitative research such as SPICE (Setting, Population, Intervention, Comparison, Evaluation). SPICE was promoted to be used for qualitative systematic reviews and was developed in the context of evidence-based librarianship. Another framework for qualitative research is ECLIPSE (Expectation, Client group, Location, Impact, Professionals, ServicE). ECLIPSE was introduced to handle health management topics. Finally, CIMO (Context, Intervention, Mechanism Outcome) is another tool for qualitative research, developed for management questions. However, none of these tools fulfills the requirements for qualitative research, since both of them have been developed for specific user groups and are not suitable and effective in more general research [49].

To summarize, PICO is a fundamental tool in EBM. It triggers researchers to define their quantitative research questions and search terms. In addition, it is the best method of question formulation to use when conducting a quantitative systematic literature review. However, it is not the optimal strategy for qualitative evidence analysis and this is the reason why new tools, like SPIDER have been proposed to compete with PICO. Nevertheless, more development and testing is necessary before SPIDER can be considered a possible and strong alternative for qualitative research.
3.3 Machine learning in EBM

As it became clear in this chapter, the medical community has a lot of information needs. To better serve those needs and any other needs that may occur from the EBM application, machine learning technologies have started to be used broadly to identify relevant key words, sentences or semantics in a given document and classify these against specified medical criteria. Such kind of information can be used in more quickly making judgements against specified PICO criteria. The broad use of PICO framework by EBM application has triggered the interest of the machine learning community to develop tools for the automatic and easier recognition of PICO components.

Demner – Fushman and Lin [50] were the first to present automatic classifiers for PICO elements. In their work, they developed a series of knowledge extractors which employ a combination of knowledge-based and statistical techniques for automatically identifying clinically relevant aspects of MEDLINE abstracts. Their goal was to identify sentences, or phrases, in abstracts relevant to each PICO element. For all the PICO elements, apart the “Outcome” element, they used the statistical classifiers to identify each element. For the “Outcome” element, they used a Naïve Bayes classifier trained with a variety of features (n-grams, positions, semantic information from MetaMap). They trained their models with a dataset consisted of 275 hand-annotated abstracts and achieved accuracies in the range of 74% - 93%.

Figure 6: Demner-Fushman and Lin’s clinical QA system architecture [50].

This work indicated that research questions regarding the role of knowledge – based and statistical techniques in advanced QA are complementary approaches and can be integrated to algorithms. It also states that basic EBM principles can be captured and implemented in a system.

Chung [51] also performed PICO classification by using CRFs sentences referring to some of the PICO elements. Intervention, Patient and Outcome elements were automatically categorized. In order to achieve this automatic categorization, Chung combined four rhetorical roles: Aim, Methods, Results and Conclusions. She later tried to combine these roles with the PICO elements. Her assumption was that each sentence in the abstract was belonging to one of these categories. Then, she focused on categorizing one PICO class at a time.

Sentence labeling on the four rhetorical roles achieved f1 – scores from 93% - 98%. Furthermore, sentences can be automatically labeled for Intervention, Participant, Outcome and Measures, in unstructured and structured abstracts where the section headings do not specifically indicate these three topics. F1- scores of up to 83% and 84% are obtained for Intervention and Outcome Measure
sentences. Unfortunately, there are two limitations to this approach: First, the overall classification performance across medical tags is not known and second, sentences are forced to always have one semantic tag.

It is worth to be mentioned that identifying PICO elements in a text is a difficult task. Boudin et al. [52] highlighted this difficulty and proposed a location-based information retrieval weighting strategy, motivated by the distribution of PICO elements. They extracted the PICO information from the query and applied a weighting model based on this information. But their annotation was based on an open text and did not take into consideration sentence boundaries which led to disagreement between the annotators. Furthermore, their classification model was trained using the headings of structured abstracts, without human annotation and this resulted in the insertion of noise into the classification model which ended in poor results.

Other work on sentence classification has also focused on rhetorical role classification, which aims at identifying the roles of sentences in text (e.g. Motivation, Result, etc.). Training and test data for this task is easy to obtain from structured scientific abstracts, which provide section headings. With respect to feature representations, previous work has relied mostly on contextual features, such as n-grams and words in specific locations. Heuristics derived from sequential features of abstracts, such as relative location of sentences and section headings have recently been explored. In terms of finding suitable machine learners, well-known machine learning techniques have been applied to the tasks, including Hidden Markov Models (HMM). Xu et al. [53] developed a novel automated approach to structuring Randomized Control Trials (RCT) abstracts by combining text classification and Hidden Markov Modeling (HMM) techniques. The results (precision of 0.94, recall of 0.93) of their approach are a significant improvement over previously reported work on automated sentences categorization in RCT abstracts.

Kim et al. [54] proposed in their research a classifier that performs two main tasks. In the first task, the algorithm identifies the key sentences in an abstract, filtering out those that do not provide the most relevant information. In the second task, it classifies sentences according to medical tags (based on the PICO criteria) used by their medical research partners. These two tasks are projected into a $(N + 1)$ way classification task, with $N$ semantic labels for key sentences and 1 label (i.e. Other) for labeling non-key sentences. For this purpose, they constructed a corpus of 1,000 medical abstracts annotated by hand with specified medical categories (e.g. Intervention, Outcome). They explored the use of various features based on lexical, semantic, structural, and sequential information in the data, using Conditional Random Fields (CRF) for classification. For the classification tasks over all labels, they achieved micro-averaged f1-scores of 80.9% and 66.9% over datasets of structured and unstructured abstracts respectively, using sequential features. In labeling only the key sentences, they reached f1-scores of 89.3% and 74.0% over structured and unstructured abstracts respectively, using the same sequential features. The results over an external dataset were lower reaching f1-scores of 63.1% for all labels, and 83.8% for key sentences.

Kiritchenko et al. [55] presented an automatic information extraction system, the ExaCT system. ExaCT helps users with locating and extracting key trial characteristics (e.g., eligibility criteria,
sample size, drug dosage, outcomes) from full-text journal articles reporting on randomized controlled trials (RCTs). The system consists of two main parts:

- An information extraction (IE) engine that scans the article for text fragments that best describe the trial characteristics,
- A web browser–based user interface (UI) that permits reviewers to review and modify the suggested selections.

The IE engine uses a statistical text classifier to locate the sentences that have the highest probability of describing a trial characteristic. This belongs to the first stage of the IE engine. The second step applies simple rules to the sentences selected from the first stage in order to extract text fragments containing the target answer. The same approach is used for all 21 trial characteristics selected for this study.

For the evaluation of the ExaCT system, authors used 50 unseen articles describing RCTs. The first stage of the IE engine, the text classifier, was able to recover 88% of the relevant sentences among its top five candidates with the top most candidate being relevant in 80% of cases. Precision and recall of the extraction rules from the second stage, were 93% and 91%, respectively. Together, the two stages of the IE engine were able to provide correct solutions in 992 out of 1050 test tasks (94%), with a majority of these (696 tasks) representing fully correct and complete answers. These experiments confirmed the efficacy of ExaCT. In addition, they demonstrated that combining a statistical method with ‘weak’ extraction rules can identify a variety of study characteristics. The system is flexible and can be extended to handle other characteristics and document types (e.g., study protocols).

Apart from the ExaCT system, Summerscales et al. [56] present the ACRES system. Their system automatically extracts necessary information from research abstracts of RCTs. These information are descriptions of the treatment groups and outcomes, as well as various associated quantities and calculates the summary statistics for every given abstract.

The ACRES system can be integrated into physician support systems and medical information retrieval systems. The evaluation of the system took place in a sample corpus of 263 British Medical Journal (BMJ) abstracts obtained via PubMed. Their system reached an f1-score of 90% of calculating summary statistics on annotated data. Overall, their system gets decent precision, at 82%. ACRES system is the first attempt at extracting outcome numbers and associating mentions with quantities for the purpose of calculating summary statistics and results are very promising.

Alamri and Stevenson [57], [58] in their studies propose a methodology that offers the ability to identify contradictory claims in the biomedical literature. In their implementation, they developed a corpus which provides examples of potentially contradictory claims and demonstrates how it can be applied and identify these claims from Medline abstracts related to the topic of cardiovascular disease. A set of systematic reviews concerned with four topics in cardiovascular disease were identified from Medline and analyzed to determine whether the abstracts they reviewed contained contradictory research claims. For each review, annotators were asked to analyze these abstracts to identify claims within them that answered the question addressed in the review. The annotators
were also asked to indicate how the claim related to that question and the type of the claim. A total of 259 abstracts associated with 24 systematic reviews were used to form the corpus. Agreement between the annotators was high, suggesting that the information they provided is reliable.

Other work attempts to bypass basic extraction tasks and address more complex biomedical QA and multi-document summarization problems to support EBM. Abacha and Zweigenbaum [59], propose a semantic approach to QA based on NLP techniques that allow a deep analysis of medical questions and documents and also semantic web technologies at representation and interrogation levels. In their research they present a semantic QA system, called MEANS. Their proposed method for “Answer Search” is based on semantic search and query relaxation.

![Figure 7: Overall architecture of the QA system MEANS [59].](image)

The evaluation of the overall performance of the MEANS system is being conducted in real questions and answers extracted from MEDLINE articles. Results are satisfactory and lead to four basic conclusions: First, the proposed approach allows dealing with different types of questions, including questions with more than one expected answer type and more than one focus. Second, it allows a deep analysis for questions and corpora using different information extraction methods, third, it is based on Semantic Web technologies, which offer more expressiveness, standard formalization languages and makes our corpus and question annotations sharable through the Web and finally, the proposed approach includes a novel query relaxation method which deals with errors or weakness of NLP methods in some cases.

Mollá and Santiago-Martinez [60], highlight in their research some of the basic NLP problems and propose a multi-document query-focused summarization method that tries to tackle these problems. First, they created a corpus for the development of such multi-document query-focused summarization task. The process to build the corpus combined the use of automated extraction of text, manual annotation, and crowdsourcing to find the reference IDs. They later performed a statistical analysis of the corpus for the particular use of single-document
summarization and showed that there is still a lot of room for improvement from the current baselines.

It appears that several approaches are focusing on identifying PICO components or other elements in medical publications in order to assist medical community in identifying and dealing with a problem more quickly and effective. In this thesis, we are proposing a different approach on identifying PICO elements from medical publications with the use of neural networks. More details are going to be explained in the following chapter.
Chapter 4 Materials and Methods

This chapter is going to describe the prediction system developed on this thesis. First, there is a brief description of the dataset used in this implementation. Subsequently, the main neural networks used in the system along with Conditional Random Field approach and Attention mechanism are explained. Finally, last section on this chapter describes in detail the neural networks.

4.1 The dataset

The use of NLP methods to automate biomedical evidence extraction has attracted less attention because there was a lack in publicly available annotated datasets in order to annotate and train the models. Nye et al. [61] address this issue by introducing EBM_NLP, a new corpus to power NLP models in support of EBM.

EBM-NLP dataset comprises of ~5000 medical abstracts describing clinical trials, multiply annotated in detail with respect to characteristics of the underlying trial Populations (e.g., diabetics), Interventions (e.g., insulin), Comparators (e.g., placebo) and Outcomes (e.g., blood glucose levels). These key informational pieces, as already known, are referred to as PICO elements.

4.1.1 Data collection

Originally, a variety of abstracts were recovered from PubMed which supports access to MEDLINE database. This database provides information regarding titles, abstracts and meta-data for articles from selected medical journals dating back to the 1970s. MEDLINE indexes over 24000 million abstracts. The majority of these abstracts have been manually assigned metadata which were used in order to select and retrieve a set of 5000 articles describing Randomized Control Trials (RCTs) with an emphasis on cardiovascular diseases, cancer and autism.

The annotation process regarding these data was divided in two steps:

1. Label acquisition by demarcating spans in the text describing the clinically abstract elements: Population, Intervention, Comparators and Outcomes. Intervention and Comparators fell into a single category. These resulted in PIO elements instead of PICO.
2. More granular annotation on the spans from step 1, by workers.

Annotations for each P, I, O element were collected individually to avoid the cognitive load imposed by switching between label sets. Annotations took place from two different set of workers, non-expert workers and medical professionals.

*Non-expert workers*

All workers participated in the annotation task were required to have an overall job approval rate of at least 90%. Each job presented to the workers required the annotation of three randomly selected abstracts from the pool of documents. Workers who were not following the provided instructions were blocked and only the best continued with the annotation task. For each abstract,
annotations received from three different workers to enable robust inference of reliable labels from noisy data.

**Expert workers**

To supplement the above data collection from non-expert workers, annotations for 200 abstracts for each PIO element from workers with advanced medical training were collected. These annotations from expert workers, are meant to serve as reference annotations. For the initial span labeling task, two medical students from the University of Pennsylvania and Drexel University provided the reference labels. In order to validate students’ annotations, three additional medical professionals were hired.

For each PIO element, all abstracts were annotated with the following four types of information:

- **Spans**: exhaustive marking of text spans containing information relevant to the respective PIO categories (Stage 1 annotation).
- **Hierarchical labels**: assignment of more specific labels to subsequences comprising the marked relevant spans (Stage 2 annotation)
- **Repetition**: grouping of labeled tokens to indicate repeated occurrences of the same information (Stage 2 annotation)
- **MeSH terms**: assignment of the metadata MeSH terms associated with the abstract to labeled subsequences (Stage 2 annotation).

**Spans**

Workers were asked to read every abstract and highlight all spans for every P, I, O element. Non-expert annotations were collected from 579 workers for 5000 abstracts and expert annotations were collected for 200 abstracts from medical students. Quality of the annotations was evaluated with the use of three evaluation strategies: simple majority vote, Dawid-Skene model [56] and HMM-Crowd [63].

**Hierarchical labels**

For each P, I, O category hierarchy of labels was developed that aimed to capture important sub categories. At this stage of annotation process, non-expert workers were presented with abstracts in which relevant spans were highlighted, based on the annotations collected in the first annotation phase. This two-step approach served two purposes:

(i) increasing the rate at which workers could complete tasks,
(ii) improving recall by directing workers to all areas in abstracts where they might find the structured information of interest.

Each one of the expert workers annotated 200 documents and reported that spans sufficiently captured the target information.

**Repetition**

Medical abstracts often mention the same information in multiple places. It is important to be able to recognize new information, especially in cases such as complex interventions, distinct measured
outcomes, or multi-armed trials. After the completion of abstract spans, workers were asked to group together sub-spans that were instances of the same information. The labels assigned by workers produced repetition labels but a more sophisticated notion of co-reference was required to identify which tokens correctly represented the entity contained in the span, and which tokens were noise.

A large majority of starting spans only contained a single target relevant to the sub-span labeling task, so identifying repetition between the starting spans is sufficient. For example, consider a starting intervention span “underwent conventional total knee arthroplasty”, there is only one intervention in the span but some annotators assigned the SURGICAL label to all five tokens while others opted for only “total knee arthroplasty.” By analyzing repetition at the level of the starting spans, agreement can be established without concern for the confounds of slight misalignments or differences in length of the sub-spans.

MeSH terms

The National Library of Medicine preserves an extensive hierarchical ontology of medical concepts called Medical Subject Headings (MeSH terms). This is part of the overarching Metathesaurus of the Unified Medical Language System (UMLS). Personnel at the NLM manually assign citations (article titles, abstracts and meta-data) indexed in MEDLINE relevant MeSH terms. These terms have been used extensively to evaluate the content of articles, and are frequently used to facilitate document retrieval.

In the case of randomized controlled trials, MeSH terms provide structured information regarding key aspects of the underlying studies, ranging from participant demographics to methodologies. A drawback to these annotations, however, is that they are applied at the document (rather than snippet or token) level. To capture where MeSH terms are instantiated within a given abstract text, a list of all terms associated with said article and instructed was provided to workers to select the subset of these that applied to each set of token labels that they annotated.

MeSH terms are domain specific and many require a medical background to understand, thus rendering this facet of the annotation process particularly difficult for untrained non-expert workers. Perhaps surprisingly, several workers voluntarily mentioned relevant background training; our pool of workers included (self-identified) nurses and other trained medical professionals. A few workers with such training stated this background as a reason for their interest in our tasks.

Of the 6963 unique MeSH terms occurring in the set of abstracts, 87% of them are only found in 10 documents or fewer and only 2.0% occur in at least 1% of the total documents.

4.1.2 Data

The exact number of medical abstracts in EBM_NLP dataset is 4993. Medical abstracts in this dataset are describing clinical trials, annotated in detail with Participants, Intervention and Outcome (PIO) elements. It is worth to be mentioned that Intervention/Comparison are treated as once. Hence, instead of four entities of the PICO model, there are three entities resulting in the PIO model. In addition, in EBM – NLP dataset apart from Participants, Intervention and Outcome elements, there is also the None entity, describing the words that do not map in any of the aforementioned PIO entities. Training labels are sourced from non-expert workers and aggregated to reduce noise. Test labels are collected from medical professionals.
The provided dataset is organized in the following files:

**Documents**: Raw text of abstracts. Documents are labeled by their PubMed identification number, e.g.:

“A double-blind crossover comparison of pindolol, metoprolol, atenolol and labetalol in mild to moderate hypertension. This study was designed to compare in a double-blind randomized crossover trial, atenolol, labetalol, metoprolol and pindolol. Considerable differences in dose (atenolol 138 +/- 13 mg daily; labetalol 308 +/- 34 mg daily; metoprolol 234 +/- 22 mg daily; and pindolol 24 +/- 2 mg daily were required to produce similar antihypertensive effects. The overall incidence of side-effects was similar with atenolol, metoprolol and pindolol but was slightly less with labetalol. Sleep disturbances and abnormal dreaming patterns were most frequent with pindolol. There was a significantly greater fall in pulse rate during atenolol and metoprolol treatment periods.”

**Tokens**: Tokenized text to which labels are assigned, e.g.:

“A double-blind crossover comparison of pindolol.”

**Part Of Speech (POS) tags**: POS tags corresponding to every token, e.g.:

“DT JJ NN NN IN NN .”

**Annotations**: Each document is multiply-annotated, so two versions of the data are presented.

- Aggregated annotations: One set of labels per document derived from a voting strategy. These annotations are recommended by the authors, since they are less noisy so more efficient for use.
- Individual: All labels from each worker.

Inside the above folders, annotations are separated by two annotation phases:

1. Starting spans: 1st phase annotations where workers highlighted spans containing target information.
2. Hierarchical spans: 2nd phase annotations with more specific label assignments.

After this, above files are separated by PICO elements and train/test partitions.
- Gold: Annotations collected by expert workers (true target test set).
• Crowd: Annotations collected by non-expert workers.

Apart from the files above, three more files are generated, in order for the data to be in a more structured form which will be easier to use. The first file is for training, the second for validation and the third one is for testing the proposed implementation in this project. All of these files have the following form: \((\text{word}, \text{POS tag}, \text{entity tag})\). An example is displayed below:

“Aerosolized \text{NNP} 1\_i
pentamidine \text{NN} 1\_i
as \text{IN} N
primary \text{JJ} N
prophylaxis \text{NN} N
for \text{IN} N
\text{Pneumocystis} \text{NNP} 1\_o
carinii \text{NN} 1\_o
\text{pneumonia} \text{NN} 1\_o”

The above three files are organized in sentences, for more flexibility. The whole corpus comprises of 53397 sentences, 41001 of which belong to the train set, 10320 to the validation set and finally, 2076 belong to the test set. These sentences contain 53445 different words, 47 different POS tags and four different entity tags: \{N, 1\_i, 1\_p, 1\_o\}.

4.2 Theoretical Background

This section describes the theoretical concepts used in the proposed implementation. Following in this section there is a detailed description of recurrent neural networks and how they work and especially LSTM and BiLSTM networks that are a specific category of recurrent neural networks that will be used in the implementation of the network architecture in this thesis. Moreover, there is a description of the Conditional Random Field theory, the Attention mechanism and an explanation of the embeddings and how important they are in word representations.

4.2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) is a class of networks that can predict the future. For example, they can analyze time series data such as stock prices and propose the best time to sell or buy. In another case, in autonomous driving systems they can anticipate car trajectories and help avoid accidents. RNNs can work on sequences of arbitrary lengths, rather than on fixed-sized inputs like other NNs. They can take sentences, documents or audio samples as input making them extremely useful for NLP systems.

A recurrent neural network looks very much like a feedforward neural network, except it also has connections pointing backward. The simplest possible RNN, composed of just one neuron receiving inputs, producing an output and sending that output back to itself is shown in Figure 8 (left). At each time step \(t\) (also called a frame), this recurrent neuron receives the inputs \(x(\text{ti})\) as well as its own output from the previous time step, \(y(t-1)\). The network representation is shown in Figure 8 (right).
It is easy to create a layer of recurrent networks. At each time step $t$, every neuron receives both the input vector $x(t)$ and the output vector from the previous time step $y(t-1)$, as shown in Figure 9.

Each recurrent neuron has two sets of weights: one for the inputs $x(t)$ and the other for the outputs of the previous time step $y(t-1)$. From now on, these weights will be referred to as $w_x$ and $w_y$. Considering a whole recurrent layer and not just one neuron, all the weight vectors can be placed in two weight matrices, $W_x$ and $W_y$. The output vector of the whole recurrent network can be computed as shown in Equation (16):

$$y(t) = \varphi(W_x^T x(t) + W_y^T y(t-1) + b)$$  \hspace{1cm} (16)

Just like in feedforward neural networks, a recurrent layer’s output can be computed in one shot for a whole mini-batch by placing all the inputs at time step $t$ in an input matrix $X(t)$.

$$Y(t) = \varphi([X(t) \quad Y(t-1)]) W + b) with \ W = \begin{bmatrix} W_x \\ W_y \end{bmatrix}$$  \hspace{1cm} (17)

- $Y(t)$ is an $m \times n_{neurons}$ matrix containing the layer’s outputs at time step $t$ for each instance in the mini-batch ($m$ is the number of instances in the mini-batch and $n_{neurons}$ is the number of neurons).
- $X(t)$ is an $m \times n_{inputs}$ matrix containing the inputs of all instances ($n_{inputs}$ is the number of input features).
- $W_x$ is an $n_{inputs} \times n_{neurons}$ matrix containing the connection weights for the inputs of the current time step.
- $W_y$ is an $n_{neurons} \times n_{neurons}$ matrix containing the connection weights for the outputs of the previous time step.
- $b$ is a vector of size $n_{neurons}$ containing each neuron’s bias term.
- The weight matrices $W_x$ and $W_y$ are often concatenated vertically into a single weight matrix $W$ of shape $(n_{inputs} \times n_{neurons}) \times n_{neurons}$.
- The notation $[X(t) \ Y(t-1)]$ represents the horizontal concatenation of the matrices $X(t)$ and $Y(t-1)$.

It is worth mentioning that $Y(t)$ is a function of $X(t)$ and $Y(t-1)$, which is a function of $X(t-1)$ and $Y(t-2)$, which is a function of $X(t-2)$ and $Y(t-3)$, and so on. This makes $Y(t)$ a function of all the inputs since $t = 0$.

Since the output of a recurrent neuron at time step $t$ is a function of all the inputs from previous time steps, it indicates it has a form of memory. A part of a neural network that preserves some state across time step is called a memory cell (or simply a cell). A single recurrent neuron, or a layer of recurrent neurons, is a very basic cell.

A cell’s state at time step $t$, denoted $h(t)$ (the “h” stands for “hidden”), is a function of some inputs at that time step and its state at the previous time step: $h(t) = f(h(t-1), x(t))$. Its output at time step $t$, denoted $y(t)$, is also a function of the previous state and the current inputs. In the case of the basic cells, the output is simply equal to the state, but in more complex cells this is not always the case.

In order to train an RNN the trick is to unroll it through time and then use regular backpropagation. This strategy is called backpropagation through time (BPTT).

![Figure 10: Backpropagation through time.](image)
Like in regular backpropagation, there is a first forward pass through the unrolled network, then the output sequence is evaluated using a cost function $C(Y(t_{\min}), Y(t_{\min}+1), ..., Y(t_{\max}))$ (where $t_{\min}$ and $t_{\max}$ are the first and last output time steps, not counting the ignored outputs), and the gradients of that cost function are propagated backward through the unrolled network. Finally, model parameters are updated using the gradients computed during BPTT. Note that gradients flow backward through all the outputs used by the cost function, not just through the final output. Moreover, since the same parameters $W$ and $b$ are used at each time step, backpropagation will do the right thing and sum over all time steps.

**LSTM models**

The Long Short-Term Memory (LSTM) networks are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber. Over the years, they were gradually improved by many researchers.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn. The key idea is that the network can learn what to store in the long-term state, what to throw away and what to read from it. As the long-term state $c_{(t-1)}$ traverses the network from left to right, it first goes through a forget gate, dropping some memories and then it adds some new memories via the addition operation (which adds the memories that were selected by an input gate). The result $c(t)$ is sent straight out, without any further transformation. So, at each time step, some memories are dropped and some memories are added. Moreover, after the addition operation, the long term state is copied and passed through the tanh function and then the result is filtered by the output gate. This produces the short-term state $h(t)$ (which is equal to the cell’s output for this time step $y(t)$).

![Figure 11: LSTM cell.](image-url)
The current input vector $x(t)$ and the previous short-term state $h_{(t-1)}$ are fed to four different fully connected layers. They all serve a different purpose:

- The main layer is the one that outputs $g(t)$. It has the usual role of analyzing the current inputs $x(t)$ and the previous (short-term) state $h_{(t-1)}$. In a basic cell, there is nothing else than this layer, and its output goes straight out to $y(t)$ and $h(t)$. In contrast, in an LSTM cell this layer’s output does not go straight out, but instead it is partially stored in the long-term state.

- The three other layers are gate controllers. Since they use the logistic activation function, their outputs range from 0 to 1. These outputs are fed to element-wise multiplication operations, so if they output 0s, they close the gate and if they output 1s, they open it. More specifically:
  - The forget gate (controlled by $f(t)$) controls which parts of the long-term state should be erased.
  - The input gate (controlled by $i(t)$) controls which parts of $g(t)$ should be added to the long-term state.
  - The output gate (controlled by $o(t)$) controls which parts of the long-term state should be read and output at this time step (both to $h(t)$ and $y(t)$).

An LSTM cell can learn to recognize an important input (that is the role of the input gate), store it in the long-term state, learn to preserve it for as long as it is needed (that is the role of the forget gate) and learn to extract it whenever it is needed. This explains why they have been so successful at capturing long-term patterns in time series, long texts, audio recordings and more.

Equations below summarize how to compute cell’s long-term state, short-term state and its output at each time step for a single instance.

$$i(t) = \sigma(W_{xi}^T x(t) + W_{hi}^T h_{(t-1)} + b_i)$$
$$f(t) = \sigma(W_{xf}^T x(t) + W_{hf}^T h_{(t-1)} + b_f)$$
$$o(t) = \sigma(W_{xo}^T x(t) + W_{ho}^T h_{(t-1)} + b_o)$$
$$g(t) = \tanh(W_{xg}^T x(t) + W_{hg}^T h_{(t-1)} + b_g)$$

$$c(t) = f(t) \otimes c_{(t-1)} + i(t) \otimes g(t)$$
$$y(t) = h(t) = o(t) \otimes \tanh(c(t))$$ (18)

- $W_{xi}, W_{xf}, W_{xo}, W_{xg}$ are the weight matrices of each of the four layers for their connection to the input vector $x(t)$.
- $W_{hi}, W_{hf}, W_{ho}, W_{hg}$ are the weight matrices of each of the four layers for their connection to the previous short-term state $h_{(t-1)}$.
- $b_i, b_f, b_o, b_g$ are the bias terms for each of the four layers.
Bi-directional LSTM models

Bidirectional Recurrent Neural Networks (BRNNs), were introduced to increase the amount of input information available to the network. BRNNs connect two hidden layers of opposite directions to the same output. Based on this connection, the output layer can get information from past (backwards) and future (forward) states simultaneously [64].

In sequence tagging tasks, both future and past features can be accessed by using a bi-directional LSTM network. The idea is that two LSTM models are combined. The first one is a forward LSTM that encodes sequences from left to right. Then, separately, there is a backward LSTM with completely separate weights to the forward LSTM. The backward LSTM is doing the same thing with the forward one with the only difference that it is encoding the sequence from right to left. Each of the hidden states is computed based on the one on the right. Finally, the hidden states from both LSTMs are concatenated in order to provide the final representations.

4.2.2 Conditional Random Field theory

Conditional Random Fields (CRF) are undirected statistical graphical models, a special case of which is a linear chain that corresponds to a conditionally trained finite - state machine. Such models are well suited to sequence analysis, and CRFs in particular have been shown to be useful in part of - speech tagging [33] and named entity recognition [34]. They have also just recently been applied to the more limited task of finding gene and protein mentions [65] with promising early results.

In contrast with traditional classifiers that predict a label for a single sample without considering "neighboring" samples, a CRF can take context into account. To do so, the prediction is modeled as a graphical model, which implements dependencies between the predictions. What kind of graph is used depends on the application. For example, in natural language processing, linear chain CRFs are popular, which implement sequential dependencies in the predictions. In image
processing the graph typically connects locations to nearby and/or similar locations to enforce that they receive similar predictions.

Laferty, McCallum and Pereira [66] define a CRF model on observations $X$ and random variables $Y$ as follows: Let $G = (V,E)$ be a graph such that $Y = (Y_u)_{u \in V}$, so that $Y$ is indexed by the vertices of $G$. Then $(X,Y)$ is a conditional random field in case, when conditioned on $X$, the random variables $Y_u$ obey the Markov property with respect to the graph: $p(Y_u|X,Y_w, w \sim u)$, where $w \sim u$ means that $w$ and $u$ are neighbors in $G$.

Thus, a CRF is a random field globally conditioned on the observation $X$. Let’s denote $x = (x_1, x_2, ..., x_m)$ as the input sequence, i.e. the words of a sentence and $s = (s_1, s_2, ..., s_m)$ the sequence of output states, i.e. the named entity tags. In CRF the conditional probability can be modeled as follows: $p(s_1, ..., s_m|x_1, ..., x_m)$.

In order to do this, a feature map $\Phi(x_1, ..., x_m, s_1, ..., s_m) \in \mathbb{R}^d$ must be defined that maps an entire input sequence $x$ paired with an entire state sequence $s$ to some $d$-dimensional feature vector. Then, probability can be modeled as a log-linear model with the parameter vector $w \in \mathbb{R}^d$

$$p(s|x; w) = \frac{\exp(w \cdot \Phi(x,s))}{\sum_{s'} \exp(w \cdot \Phi(x,s'))}$$ (19)

Where $s'$ ranges over all possible output sequences.

For the estimation of $w$, let assume there is a set of $n$ labeled examples $\{(x^i, s^i)\}_{i=1}^n$. The regularized log-likelihood function $L$ is defined as:

$$L(w) = \sum_{i=1}^n \log p(s^i|x^i; w) - \frac{\lambda_2}{2} ||w||_2^2 - \lambda_1 ||w||_1$$ (20)

The terms $\frac{\lambda_2}{2} ||w||_2^2$ and $\lambda_1 ||w||_1$ forces the parameter vector to be small in the respective norm. This penalizes the model complexity and is known as regularization. The parameters $\lambda_1$ and $\lambda_2$ allow to enforce more or less regularization. The parameter vector $w^*$ is then estimated as:

$$w^* = \arg \max_{w \in \mathbb{R}^d} L(w)$$ (21)

By estimating the vector $w^*$, it is possible to find the most likely tag a sentence $s^*$ for a sentence $x$ by

$$s^* = \arg \max_s p(s|x; w^*)$$ (22)

4.2.3 Embeddings

An embedding is a small and dense vector that can represent a word in a vocabulary. It is an improvement over the traditional bag – of - word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire vocabulary. These representations were sparse because the vocabularies were vast and a given word or document would be represented by a large vector comprised mostly of zero values. Instead, in an embedding, words are represented by dense vectors where a vector represents the
projection of the word into a continuous vector space. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. The position of a word in the learned vector space is referred to as its embedding.

At the beginning of training, embeddings are chosen randomly, but during training, backpropagation automatically moves the embeddings around in a way that helps the network perform its task. This means that similar words will gradually cluster close to one another, and even end up organized in a rather meaningful way. Apart from choosing randomly the embeddings and improve them during training, word embeddings representation are already available that can save time and effort and provide better results if using them, such as Word2Vec and GloVe.

GloVe word vector representation is the one used in this implementation. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word - word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space [67]. It provides word vector representations of various words and in different dimensions.

![Figure 13: Dependencies between word vectors – GloVe.](image)

4.2.4 Attention mechanism

Attention mechanism in Deep Learning (DL) is based on the concept of directing the focus and pay greater attention to certain factors when processing the data. Attention is a component of a network’s architecture and is in charge of managing and quantifying the relation:

- Between input and output elements (General Attention),
- Within the input elements (Self-attention).
While Attention does have its application in other fields of DL such as Computer Vision, its main breakthrough and success comes from its application in NLP tasks. This is due to the fact that Attention was introduced to address the problem of long sequences in Machine Translation (MT) which is also a problem for other NLP tasks as well.

A simple example of how Attention works in Machine Translation task is the following: There is the English sentence “How was your day” which needs to be translated to its French version “Comment se passe ta journée”. The Attention component of the network, for each word in the output sentence will map the important and relevant words from the input sentence and assign higher weights to these words, enhancing the accuracy of the output prediction.

![Figure 14: Weights are assigned to input words at each step of the translation.](image)

Attention mechanism is widely used in Machine Translation tasks because it can take two sentences, turn them into a matrix where the words of one sentence form the columns, and the words of another sentence form the rows, and then make matches, identifying relevant context, as already seen in Figure 14. But usage of attention cannot be limited only to correlate meaning between sentences in two different languages. The same sentence can also be used along the columns and the rows, in order to understand how some parts of that sentence relate to others. For example, where are my pronouns’ antecedents? This is called self-attention.

A neural network with an attention mechanism knows how to disregard the noise and focus on what’s relevant, how to connect two related words that in themselves do not carry markers pointing to the other. Attention allows to look at the totality of a sentence, to make connections between any particular word and its relevant context.

RNNs have a memory problem. There’s only so much they can remember about long-range dependencies (the words they saw a long time ago that are somehow related to the next word). That is, RNNs put too much emphasis on words being close to one another, and too much emphasis on upstream context over downstream context. Attention mechanism fixes that problem.
4.3 Methodology

The implementation followed in this thesis is using a hybrid approach combining a Bi-directional LSTM – CRF (BiLSTM - CRF) model based on Zhiheng Huang [68] methodology. Different inputs to the model were tested (words, characters, POS tags) and new layers (attention mechanism) were added to the base model in order to test their efficacy.

The steps for the implementation are the following:

- Data preprocessing
- Build and train of a BiLSTM - CRF model or a BiLSTM – CRF – Attention model
- Evaluation of model
- Predictions

Below are described in detail the different steps of this methodology.

As already mentioned, entity tags are encoded using PICO framework, where each entity label is prefixed with either 1_p, 1_i, or 1_o tags. All other words, which do not refer to entities of interest, are labeled with the N tag. Each sentence in the dataset is represented as a list of tuples: [(Token_1, PoS_1, Tag_1), … , (Token_n, POS_n, Tag_n)].

4.3.1 Preprocessing

In order to preprocess data it is essential to build dictionaries of the different words, characters, POS tags and entity tags that belong to the corpus. Dictionaries are important in NER tasks because by using them it is easy to map each word, character, POS tag and entity to a corresponding integer ID. Representing words, characters, POS tags and entity tags as integers saves a lot of memory and it is a necessary step because neural network models cannot understand words but only numbers. Thus, word input needs to be converted to corresponding vectors so neural networks can process the input information. A requirement of the dictionary construction is the creation of different vocabularies of the different words, characters, POS tags and entity tags of the dataset. Vocabularies will contain all the different words in the dataset and it will be easier to map each one of them to an integer ID and store it in a dictionary.

In order to feed the text to the model, all sentences should have equal lengths. This is something that does not exist in unprocessed text. Sentences can consist of different number of words. In order to make all sentences the same length, padding is necessary. Padding is a process in which after defining the desired length of a sentence (e.g. number of words the sentence consists of), zeroes are added in the sentences that have less words than defined in order to make them equal in length with the rest sentences on the dataset.

In this specific dataset, maximum number of words was set to 100, after visualizing the number of words that belong to each sentence in the three datasets (train, validation and test) as can be seen in the figures below.
In Figure 15 it is clear that tokens per sentence come as a Gaussian distribution, which is expected due to the central limit theorem. For the train test it seems that most sentences have less than 50 tokens, but there is a significant number which contains more than 50 tokens and less than 100. Based on this diagram, 100 seems a good choice in order to keep as much information as possible from the sentences without adding too much noise. Same logic applies for Figure 16, which represents tokens per sentence for the validation dataset. In this dataset as well, tokens per sentence follow Gaussian distribution.

Figure 17 describes the token per sentence dataset on the test dataset. In this dataset, the majority of the sentences has less than 60 tokens per sentence, but there are sentences with number of tokens ranging between 80 and 100 so 100 is also a choice suited for this dataset.
The same padding applied to words was applied to tag entities and POS tags, since a POS tag and an entity tag describe each word in the dataset.

First, word dictionary was constructed. Each word was mapped to an integer ID. The index of words was increased by two to use integer “0” as a padding value and integer “1” as a value for the unknown words. In order to generate a dictionary for the characters it is important to create the sequence of characters for every token. In order to do so, the maximum number of characters in a word needs to be defined; in this case it is set to 12, considering the fact that the majority of the existing words do not exceed 12 characters. So, with this choice, some words might be cut, which is not desired, but it avoids the problem of adding noise to the character data. The index of characters was increased by one to use integer “0” as a padding value. For the construction of POS dictionary and entity tag dictionary the index was also increased by one in order to use value “0” as a padding value.

After building the different dictionaries, all words were mapped to a sequence of numbers for all three sets, all characters of every word of every sentence were mapped to a sequence of numbers for all three sets and finally, all POS tags describing the words were mapped to a sequence of numbers for all sets. These token representations of the three different inputs are going to be used in order to describe the word and contextual word representation of our model.

For the word representation, it is necessary to use a representation $w \in \mathbb{R}^n$ for each word. In order to do that some pre-trained word embeddings $w_{glove} \in \mathbb{R}^d$ were loaded. Although GloVe contains vector representations of most words, it is not enough. Especially in medical text (like in this case) it is very likely to encounter words that are not present in GloVe’s vocabulary. Such a word is called an out-of-vocabulary (OOV) word. GloVe deals with these OOV words by simply assigning them some random vector values. If not remedied, this random assignment would end up confusing the model.
Therefore, another mechanism is required, that can handle OOV words. This mechanism is the character level embeddings. Character level embeddings find numeric representations of words by looking at their character-level compositions. In English, all words are formed by 26 (or 52 if including both upper and lower case character, or even more if including special characters) characters. It is difficult to find unknown characters, like in words. Hence, it is a good fit for misspelling words or new words which is very common in text related to the medical field. So each character $c_i$ of a word $w = [c_1, ..., c_p]$ is associated to a vector $c_i \in \mathbb{R}^{d_2}$. In character embeddings case, no pre-trained method was used, so at the beginning, embeddings are chosen randomly.

Finally, for POS representation there is a vector $w \in \mathbb{R}^{d_3}$ for each word. In POS embeddings, like in character embeddings, no pre-trained method was used.

4.3.2 Models

The main model architecture used in this implementation combines a Bi-directional LSTM network with a CRF network to form a Bi-directional LSTM – CRF (BiLSTM - CRF) model. Different input data were used to test which word representation is more efficient. The BiLSTM - CRF model was later enhanced with the attention mechanism.

Model architecture can break down into three main pieces:

- Input layers,
- Main architecture,
- Prediction.

This section will first describe in detail the main layers of the BiLSTM – CRF network and later on the different embedding layers that also constitute the input layers of the network. Subsequently, it will describe the different implementations of the BiLSTM – CRF network with the addition of attention mechanism either on the main layers or the input layers of the network.

**Bi – LSTM – CRF network**

The implementation of the BiLSTM – CRF neural network consists of four main layers. An input layer, which in this implementation can vary, so at this point it is considered as one layer, a BiLSTM layer, a dense layer and finally a CRF layer.

Regarding the input layer, as already mentioned it can vary. Different inputs were tested for the BiLSTM – CRF network that are going to be described in detail later on. In order to prevent the models from depending on one representation or the other too strongly (in case of multiple inputs), a dropout layer is used before the main BiLSTM layer.

Regarding the BiLSTM layer, in sequence tagging task, it is possible to access both past and future input features for a given time with the use of a Bidirectional LSTM network as proposed in [69]. With BiLSTM networks past features (via forward states) and future features (via backward states) for a specific time frame are efficiently used. BiLSTM network is trained using backpropagation through time. The forward and backward passes over the unfolded network over time are carried out in a similar way to regular network forward and backward passes, except the need to unfold
the hidden states for all time steps. In this implementation, forward and backward passes are being done for whole sentences and the only need to reset the hidden states to zero is at the beginning of each sentence. Batch implementation enables multiple sentences to be processed at the same time. Hence, a BiLSTM network is running over the input sequence and obtains another sequence of vectors which are the concatenation of the two hidden states of BiLSTM network. The sequence of vectors that constitute the output of BiLSTM layer is connected to a dense layer in order to avoid overfitting of the model.

In order to make final prediction a CRF layer will connect to the dense layer in order to make use of the neighboring tagging decisions. Otherwise, the tagging decision is local. A CRF layer is represented by lines, which connect consecutive output layers. It has a state transition matrix as parameters and can efficiently use past and future tags to predict the current tag. This ability is similar to the ability of BiLSTM networks to use past and future input features. Let’s consider that matrix of scores \( f_\theta([x]^T) \) are output by the network. Input \([x]^T\) is dropped for notation simplification. The element \([f_\theta]_{i,t}\) of the matrix is the score output by the network with parameters \( \theta \), for the sentence \([x]^T\) and for the \(i\)-th state to \(j\)-th for a pair of consecutive time steps. It is worth mentioning that the transition matrix is position independent. The new parameters of the network can be denoted as \( \theta' = \theta \cup \{[A]_{i,j} \forall i,j \} \). The scores of a sentence \([x]^T\) along with a path of tags \([i]^T\) is then given by the sum of transition scores and network scores:

\[
s([x]^T, [i]^T, \theta') = \sum_{t=1}^{T} ([A]_{i,t-1} + f(\theta)_{[i],t,t})
\]  

(23)

Dynamic programming can then be used efficiently to compute \([A]_{i,j}\) and optimal tag sequences for inference [33], [66]. In Figure 18 below there is a display of the BiLSTM – CRF architecture.

![Figure 18: BiLSTM – CRF network architecture.](image)

**Input: Word vector representations**

In this implementation, basic word representation was inserted to the BiLSTM - CRF model. Regarding the embedding layers, a word embedding layer was used to represent words as vectors.
It has been shown in [70] that word embeddings can play a vital role to improve sequence tagging performance. Word embedding vectors were initialized with GloVe pre-trained models instead of random initialization.

The word embedding layer consists the input layer that was connected to the BiLSTM layer. This layer can efficiently use past and future input features. The BiLSTM layer runs over the sequence of word vectors and obtains another sequence of vectors (the concatenation of the two hidden states of the BiLSTM), $w_t \in \mathbb{R}^k$.

Thus, the input is a sequence of $m$ word vectors $w_1, ..., w_m \in \mathbb{R}^n$ and the output is a sequence of vectors $w_1, ..., w_t \in \mathbb{R}^k$ that capture information at the word level (syntax and semantics). At this point, each word $w$ is associated to a vector $w_t$ that captures information from the meaning of the word.

As can be seen in Figure 19, each vector $w_t$ is inserted to the BiLSTM layer in order to create the hidden states of the network and the output of the BiLSTM layer is connected to a dense layer in order to avoid overfitting. A CRF layer is connected with the output of the dense layer in order to make a prediction for every word in the dataset.

*Figure 19: BiLSTM network with word vector representations as input.*
Input: Word and character vector representations of words

In this implementation, for each word, we wanted to build a vector \( w \in \mathbb{R}^n \) that could capture the meaning and relevant features of each word. This vector will occur after the concatenation of \( w_{glove} \in \mathbb{R}^{d_1} \) from GloVe pre-trained models and of a vector that will contain extracted features from the character level \( w_{chars} \in \mathbb{R}^{d_2} \). Regarding the selection of character level features, two options were available. First, to use handcrafted features, like a component with a 0 or 1 if the word starts with a capital letter for instance, and second with the use of a neural network to make this extraction automatically. Second option was chosen as more suitable to the problem and a BiLSTM model was used at the character level.

First, a word embedding layer was used to represent words as vectors. Word embedding vectors were initialized with GloVe pre-trained models instead of random initialization.

Each character \( c_i \) of a word \( w = [c_1, \ldots, c_p] \) is associated to a vector \( c_i \in \mathbb{R}^{d_2} \). Character embedding layer is initialized with random initialization and a BiLSTM network is running over the sequence of character embeddings and concatenates the final states to obtain a fixed-size vector \( w_{chars} \in \mathbb{R}^{d_2} \). This vector captures the morphology of the word. Figure 20 shows the word level representation of a word from character embeddings with the use of a BiLSTM network.

![Figure 20: Word level representation from character embeddings.](image)

Finally, word embeddings \( w_{glove} \) are concatenating with character embeddings \( w_{chars} \) in order to get a vector representing a word \( w = [w_{glove}, w_{chars}] \in \mathbb{R}^n \), where \( n = d_1 + d_2 \). Once word representation is ready, it is inserted to a BiLSTM layer. BiLSTM is running over the sequence of word vectors and obtains another sequence of vectors (the concatenation of two hidden states),
This time, as input, there is a sequence of vectors \( h_1, \ldots, h_m \in \mathbb{R}^k \). Whereas \( w_t \) vector only captured information at the word level (syntax and semantics), vector \( h_t \) can also take context into account. Vectors \( h_1, \ldots, h_m \in \mathbb{R}^k \) are inserted into a dense layer which is connected to a CRF layer in order to predict the tag for every word. Figure 21 displays the BiLSTM – CRF network with word and character representations as input to the network.

![Diagram of BiLSTM network with word and character vector representations as input.](image)

**Figure 21**: BiLSTM network with word and character vector representations as input.

**Input: Word, character and POS vector representations of words**

In the third implementation, word representation along with contextual representation of each word will be enhanced with the POS representation. In a text, a word can have different meaning if it is used as a noun or adjective in a sentence. The way words are used in a sentence can change their meaning and importance.

First, a word embedding layer was used to represent words as vectors. Word embedding vectors were initialized with GloVe pre-trained models instead of random initialization. Second, a
A character embedding layer with a BiLSTM network was used to represent the content of each word.

A POS embedding layer was used to represent POS tags as vectors. POS embeddings were randomly initialized since there are no pre-trained models regarding POS tags. POS embedding layer is initialized according to the word embeddings layer. Each word vector $w$ is associated to a POS vector $p$.

Finally, word embeddings $w_{glove}$ are concatenating with character embeddings $w_{chars}$ and POS embeddings $w_{POS}$ in order to get a vector representing a word $w = [w_{glove}, w_{chars}, w_{POS}] \in \mathbb{R}^n$, where $n = d_1 + d_2 + d_3$.

![BiLSTM network with word, character and POS vector representations as input.](image)

*Figure 22: BiLSTM network with word, character and POS vector representations as input.*

As it can be seen in Figure 22, when the new word representation is ready, it is inserted to the BiLSTM network, like in the previous two cases, the BiLSTM network is then connected to a
dense layer and finally, the output of this layer is inserted to the CRF layer in order to make the final predictions.

**Bi–LSTM – CRF – Attention network**

The main BiLSTM – CRF network was enhanced with an attention layer in order to test if this layer would improve the performance of the model or not. The architecture of the model remains the same, although the attention mechanism was tested in different parts of the model.

As already mentioned, attention mechanism was originated in Machine Translation problems. MT systems comprise of a sequence – to - sequence encoder and decoder. Semantics of a sentence is mapped into a fixed - length vector representation by an encoder, and then the translation is generated based on that vector by a decoder. In the original MT model, the decoder generates a translation solely based on the last hidden state. But it is somewhat unreasonable to assume all information about a potentially very long sentence can be encoded into a single vector, and that the decoder will produce a good translation solely based on that. With an attention mechanism, instead of encoding the full source sequence into a fixed - length vector, we allow the decoder to attend to different parts of the source sentence at each step of the output generation. The model learns what to attend to based on the input sentence and what it has produced so far.

In this implementation the idea is similar. An attention layer with an attention matrix $A$ captures the similarity of any token with respect to all the neighboring tokens in an input sequence. The element $a_{t,t'} \in A$ captures the similarity between the hidden state representations $h_t$ and $h_{t'}$ of tokens $x_t$ and $x_{t'}$ at time steps $t$ and $t'$respectively. The attention mechanism is implemented similar to a BiLSTM cell as follows:

$$g_{t,t'} = \tanh(W_g h_t + W_g' h_{t'} + b_g),$$

$$a_{t,t'} = \sigma(W_a g_{t,t'} + b_a),$$

Where $\sigma$ is the element wise sigmoid function, $W_g$ and $W_{g'}$ are the weight matrices corresponding to the hidden states $h_t$ and $h_{t'}$; $W_a$ is the weight matrix corresponding to their non-linear combination; $b_g$ and $b_a$ are the bias vectors.

The attention focused hidden state representation $l_t$ of a token at timestep $t$ is given by the weighted summation of the hidden state representation $h_{t'}$ of all other tokens at timesteps $t'$ and their similarity $a_{t,t'}$ to the hidden state representation $h_t$ of the current token. Essentially, $l_t$ dictates how much to attend to a token at any timestep conditioned on their neighborhood context. This can be used to highlight the model’s final tagging decision based on token importance.

$$l_t = \sum_{t'=1}^{n} a_{t,t'} \cdot h_{t'}$$

First, the attention mechanism was applied in POS embeddings, second approach applied a simple attention layer after the main BiLSTM layer and finally, attention mechanism was applied in word and character embeddings and POS tag embeddings separately before the concatenation of the attention vectors. These three implementations are describe in more details below.
Input: Word and character vector representation and POS attention vector representation

In this implementation, attention mechanism was used in the POS tags embeddings. The output of the POS embedding layer goes as input to the attention layer that learns which vectors to focus or attend to in particular, generating the attention vector $a_{POS} \in A$. Figure 23 shows the attention mechanism on POS embeddings.

![Figure 23: Attention mechanism on POS embeddings.](image)

The first layer is the word embedding layer, $w_{glove}$, the second layer is the character embedding layer $w_{chars}$ and the third layer is the POS embedding layer $w_{POS}$. As already described, attention mechanism is applied in POS embeddings resulting in an attention vector $a_{POS}$. This vector is used along with $w_{glove}$ and $w_{chars}$ in order to represent a word $w = [w_{glove}, a_{POS}, w_{chars}] \in \mathbb{R}^n$, where $n = d_1 + d_2 + A$. 
Each word representation $w$ is used as input to the BiLSTM layer that generates its hidden state representation $h_t$ as a concatenation of the forward and backward LSTM states. The hidden states of BiLSTM are input features to a dense layer which is connected to a CRF layer that enforces tagging consistency considering dependency between output tags and the hidden state representation of tokens at each timestep.

**Input:** Word and character attention vector representation and POS attention vector representation

In the second implementation with the BiLSTM – CRF model with the attention mechanism, attention mechanism is applied on the concatenation of words and character embeddings and on POS embeddings. In the model architecture, the first layer is the word embeddings, $w_{glove}$ and the second layer is the character embeddings $w_{chars}$. These two vectors are concatenating and their
concatenation is inserted into an attention layer as displayed in Figure 25. The third layer is the output of the attention layer, an attention vector $a_{glove,chars} \in A_1$.

Next, the output of the POS embedding layer, which constitutes the fourth layer of the model goes as input to the attention layer as well, generating a second attention vector $a_{POS} \in A_2$. These two attention vectors are concatenating resulting in $a = [a_{glove,chars}, a_{POS}] \in A$, where $A = A_1 + A_2$.

This vector is then used as input to the BiLSTM layer and the hidden states generated by BiLSTM are used as input to a dense layer and finally the output of the dense layer is inserted into a CRF layer in order to make the final predictions.

Input: Word, character vector representation and POS vector representation with attention in BiLSTM

Finally, the third implementation regarding attention mechanism is using the attention layer after the main BiLSTM layer. Figure 26 describes the architecture in details. The first layer is the word embedding layer, $w_{glove}$, the second layer is the character embedding layer $w_{chars}$ and the third layer is the POS embedding layer $w_{POS}$. Word embeddings $w_{glove}$ are concatenating with character embeddings $w_{chars}$ and POS embeddings $w_{POS}$ in order to get a vector representing a word $w = [w_{glove}, w_{chars}, w_{POS}] \in \mathbb{R}^n$, where $n = d_1 + d_2 + d_3$.
This vector representation is used as input to the BiLSTM layer and generates the hidden states. The output of BiLSTM goes as input to the attention layer that learns which states to focus or attend to in particular — generating the attention-focused hidden state representation $< l_t >$ for the input sequence. These representations are inserted to a dense layer, to avoid overfitting and then used as input features in the CRF that enforces tagging consistency, considering dependency between output tags and the hidden state representation of tokens at each timestep.

Figure 26: BiLSTM – CRF with attention layer.
4.3.3 Experimental setup and training procedure

Regarding embedding layers, for word embeddings GloVe pre-trained models with 50 and 200 dimensions were tested. Finally, the 200 dimensions of GloVe embeddings were used since they performed better than the GloVe embeddings of 50 dimensions. Regarding character embeddings, vectors with 100 dimensions were used and for POS embeddings, vectors with 50 dimensions were used. The BiLSTM network for character embeddings uses forward and backward LSTMs whose dimensions are set to 100. In order to prevent the models from depending on one representation or the other too strongly, a dropout training with a probability of 50% is used to improve the generalization performance.

For all models, presented networks are trained by using back-propagation algorithm, updating the parameters on every training example, one at a time, using the Rmsprop optimizer with a learning rate of 0.001. Several other optimizers were tested, such as Adam and SGD, but Rmsprop reported the best performance.

BiLSTM – CRF models use a single layer for the forward and backward LSTMs whose dimensions are set to 200. In total 100 epochs were used for training the models. During training, early stopping method was applied, and set on 5 epochs, which allows the specification of an arbitrary large number of training epochs and stop training once the model performance stops improving on a holdout validation dataset. For all models, no more than 15 epochs were necessary for training, for the networks to achieve their best performance.

Each epoch divides the whole training data to batches and processes one batch at a time. Each batch contains a list of sentences which is determined by the parameter of batch size. In the experiments, a batch size of 32 samples was tested. Selected batch size means to include sentences whose total length is no greater than the selected batch size. For each batch, it first runs a BiLSTM – CRF model with a forward pass which includes the forward pass for both forward state and backward state of LSTM. As a result, it occurs the output score $f_\theta([x]^T)$ for all tags at all positions. Then a dense layer gets all these scores with 400 nodes, same dimensions as set to BiLSTM. Then a CRF layer is running with five output nodes, as much as the desired predicted labels plus the PAD layer used for the padded elements in the sentences. CRF layer runs a forward and backward pass to compute gradients for network output and state transition edges. After that, errors are back propagated from the output to the input, which includes the backward pass for both forward and backward states of LSTM. Finally, network parameters are updated which include the state transition matrix $[A]_{i,j} \forall i,j$, and the original BiLSTM parameters $\theta$.

4.3.4 Evaluation and prediction

An important part on every neural network architecture is the proper evaluation of the model. This can be a complex problem in a sequence model with token-based labels because some tag entities may occur more often than others and in addition some tag entities may span multiple tokens. The first problem can be solved by using the right metrics for the evaluation while the second problem can be solved by aggregating the token level predictions in the right way.

Accuracy is a metric used very often in classification problems. Although in sequence tagging problems is not a suitable metric to use since it can be misleading. Let’s assume that label “N” is
the most common label in the dataset. A model that would always predict the label “N” could achieve very high accuracy, even 95% or higher, but the model wouldn’t learn anything.

Accuracy score is defined as:

\[
\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}
\]  
(27)

To deal with this problem other metrics such as F1, precision and recall are more appropriate for this kind of problems.

Precision score is defined as:

\[
\text{Precision} = \frac{TP}{TP+FP}
\]  
(28)

Recall score is defined as:

\[
\text{Recall} = \frac{TP}{TP+FN}
\]  
(29)

F1 score is defined as the harmonic mean of precision and recall:

\[
F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\]  
(30)

The different implementations of the BiLSTM – CRF model and the BiLSTM – CRF models with attention mechanism are tested in the test set using the above three metrics. Accuracy metric is also reported for completeness.

Apart from classification metrics, loss metrics are also a good way to evaluate the performance of a neural network model. Neural networks are trained using stochastic gradient descent and require the choice of a loss function when designing and configuring the model. There are many loss functions to choose from and it can be challenging to know what to choose, or even what a loss function is and the role it plays when training a neural network.

A neural network learns to map a set of inputs to a set of outputs from training data. It is very difficult to calculate the perfect weights for a neural network since there are too many unknowns. Instead, the problem of learning is cast as a search or optimization problem and an algorithm is used to navigate the space of possible sets of weights the model may use in order to make good or good enough predictions. Typically, a neural network model is trained using the stochastic gradient descent optimization algorithm and weights are updated using the backpropagation of error algorithm. The “gradient” in gradient descent refers to an error gradient. The model with a given set of weights is used to make predictions and the error for those predictions is calculated. The gradient descent algorithm seeks to change the weights so that the next evaluation reduces the error, meaning the optimization algorithm is navigating down the gradient (or slope) of error.

In the context of an optimization algorithm, the function used to evaluate a candidate solution (i.e. a set of weights) is referred to as the objective function. Usually the purpose is to maximize or minimize the objective function, meaning that it is necessary to find a candidate solution that has the highest or lowest score respectively. Typically, with neural networks, the purpose is to
minimize the error. As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply “loss”. The cost or loss function has an important job in that it must faithfully distill all aspects of the model down into a single number in such a way that improvements in that number are a sign of a better model.

In calculating the error of the model during the optimization process, a loss function must be chosen. For this implementation, log-likelihood function was chosen as the loss function.

In statistics, the likelihood function measures the goodness of fit of a statistical model to a sample of data for given values of the unknown parameters. It is formed from the joint probability distribution of the sample, but viewed and used as a function of the parameters only, thus treating the random variables as fixed at the observed values. The likelihood function describes a hypersurface whose peak, if it exists, represents the combination of model parameter values that maximize the probability of drawing the sample obtained [71]. The procedure for obtaining these arguments of the maximum of the likelihood function is known as maximum likelihood estimation, which for computational convenience is usually done using the natural logarithm of the likelihood, known as the log-likelihood function. Additionally, the shape and curvature of the likelihood surface represent information about the stability of the estimates, which is why the likelihood function is often plotted as part of a statistical analysis [72].

Consider a set of $m$ examples $X = \{x^{(1)}, ..., x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(x)$.

Let $p_{model}(x; \theta)$ be a parametric family of probability distributions over the same space indexed by $\theta$. In other words $p_{model}(x; \theta)$ maps any configuration $x$ to a real number estimating the true probability $p_{data}(x)$.

The maximum likelihood estimator for $\theta$ is defined as:

$$\theta_{ML} = \arg_{\theta} \max p_{model}(X; \theta),$$

$$= \arg_{\theta} \max \prod_{i=1}^{m} p_{model}(x^{(i)}; \theta)$$

(31)

This product over many probabilities can be inconvenient for various reasons. For example, it is prone to numerical underflow. To obtain a more convenient but equivalent optimization problem, we observe that taking the logarithm of the likelihood does not change its $\text{argmax}$ but does conveniently transform a product into a sum:

$$\theta_{ML} = \arg_{\theta} \max \sum_{i=1}^{m} p_{model}(x^{(i)}; \theta)$$

(32)
Chapter 5 Results

This chapter presents the results of the developed neural network models. It presents the classification metrics used for the evaluation of the models (F1 score, precision, recall) in the tables below. Every name entity (Participant, Intervention, Outcome, None) is characterized by a value of each metric and finally there is a mean value for every metric across all name entities. Apart from F1 score, precision and recall metrics there are also diagrams that indicate the evolution of the accuracy and loss metric conducted in each epoch for every model and finally, accuracy results and loss metric on the test dataset are also stated.

Table 1 indicates the evaluation results for every name entity for the first BiLSTM – CRF model that uses word vector representation as an input to the neural network. It appears that the best F1-score is achieved for the name entity Participant and reaches a value of 51\%, while the lowest is achieved for the name entity Outcome with a value of 29\%. The mean F1-score for this model across all entities is 38\%.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.47</td>
<td>0.35</td>
<td>0.40</td>
<td>5057</td>
</tr>
<tr>
<td>Participant</td>
<td>0.54</td>
<td>0.48</td>
<td>0.51</td>
<td>663</td>
</tr>
<tr>
<td>Intervention</td>
<td>0.47</td>
<td>0.27</td>
<td>0.34</td>
<td>1674</td>
</tr>
<tr>
<td>Outcome</td>
<td>0.34</td>
<td>0.25</td>
<td>0.29</td>
<td>1852</td>
</tr>
<tr>
<td>Average</td>
<td>0.45</td>
<td>0.32</td>
<td>0.38</td>
<td>9246</td>
</tr>
</tbody>
</table>

Figure 27 below displays the accuracy values during training for the train and validation dataset for every epoch.

As it can be seen in the diagram, accuracy on train set follows an upward course. In the first epoch, accuracy has a value of 71.39\% and in the last epoch the value is 82.94\%. Although, in validation set, it does not follow an upward course in every epoch. It is clear that in the second epoch there is a big fall on the accuracy value, probably because the model did not have the time to learn the data on the first epoch and trained efficiently in only one epoch. In the first epoch accuracy has a value of 73.98\% and in the last epoch it has a value of 78.08\%.

This behavior on both accuracy curves (for train and validation sets) is expected. It indicates that the model is learning from the data but not with a high rate and also it is safe to conclude that it is not overfit the train data. For the test data, accuracy score reaches the value of 79.2\%.
The diagram on Figure 28 displays the loss of the model for every epoch on both train and validation set. The loss metric in both sets decreases. In the train set, there is a big fall on the loss curve between the first two epochs which indicates that at the beginning, the model was assigning random values to the weights. After the second epoch though, the curve is decreasing more smoothly. On the validation set, this fall does not happen, because the model has already calculated the weights from the train data. On the train set, the final loss value is 7.2291, while in validation set it is 7.3052. Finally, in test set loss metric is 7.2207.
Figure 28: Loss metric for train and validation dataset for every epoch during training of the neural network. X-axis presents the number of epochs, while y-axis presents the loss values.

Table 2 indicates the evaluation results for every name entity for the second BiLSTM–CRF model that uses word and character vector representations as an input to the neural network. It appears in this model as well, that the best F1-score is achieved for the name entity Participant and reaches a value of 54%, while the lowest is achieved for the name entity Outcome with an F1 score of 33%. The mean F1-score for this model is 38%.

Table 2: Classification report of BiLSTM – CRF network with word and character vectors as input.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.46</td>
<td>0.33</td>
<td>0.38</td>
<td>5057</td>
</tr>
<tr>
<td>Participant</td>
<td>0.58</td>
<td>0.50</td>
<td>0.54</td>
<td>663</td>
</tr>
<tr>
<td>Intervention</td>
<td>0.57</td>
<td>0.27</td>
<td>0.36</td>
<td>1674</td>
</tr>
<tr>
<td>Outcome</td>
<td>0.46</td>
<td>0.26</td>
<td>0.33</td>
<td>1852</td>
</tr>
<tr>
<td>Average</td>
<td>0.48</td>
<td>0.32</td>
<td><strong>0.38</strong></td>
<td>9246</td>
</tr>
</tbody>
</table>

Figure 29 below displays the performance learning curve of accuracy values during training for the train and validation set for every epoch.

As it can be seen in the diagram, accuracy on train set follows an upward course. In the first epoch, accuracy has a value of 73.27% and in the last epoch the value is 83.77%. Although, in validation set, its behavior is not the one expected since it does not follow an upward course in every epoch. There is a big fall towards the end, probably because the model is close to overfitting, but then validation accuracy increases again so the model does not overfit the train data. Although these fluctuations on the validation accuracy curve indicate that for some epochs, the model is randomly
assigning labels to the instances. This behavior probably means that the model is using a random sample from the validation dataset; the validation set at each evaluation step is different. This would be expected and justified in the first epochs, but such kind of behavior on the last epochs is hard to explain. In the first epoch accuracy has a value of 76.49% and in the last epoch it has a value of 79.82%. This value is not very different from 83.77%, the train validation on the last epoch. For the test data, accuracy score reaches the value of 81%.

![Accuracy results on train and validation dataset for every epoch during training of the neural network.](image)

*Figure 29: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values.*

The diagram on Figure 30, displays the loss of the model for every epoch on both train and validation set. The loss metric in both sets decreases with loss on train dataset to be 7.2291 and in validation dataset to be 7.2094. Finally, in test dataset loss metric is 7.3015.

From the diagram, it is clear that for train data, loss is stabilized in the value of 7.2291, which means that the model cannot extract more information from the data. Same conclusion can occur regarding validation data, since the loss curve for this dataset also appears to be stable on 7.2094 value for the last epochs.
Table 3 indicates the evaluation results for every name entity for the third BiLSTM – CRF model that uses word, character and POS vectors as an input to the neural network. It appears that the best F1-score is achieved again for the name entity *Participant* and reaches a value of 50%, while the lowest is achieved this time for the name entity *Intervention*, with a value of 29%. The mean F1-score for this model is 34%.

### Table 3: Classification report of BiLSTM – CRF network with word, character and POS tag vectors as input.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>None</strong></td>
<td>0.42</td>
<td>0.28</td>
<td>0.34</td>
<td>5057</td>
</tr>
<tr>
<td><strong>Participant</strong></td>
<td>0.51</td>
<td>0.50</td>
<td>0.50</td>
<td>663</td>
</tr>
<tr>
<td><strong>Intervention</strong></td>
<td>0.52</td>
<td>0.20</td>
<td>0.29</td>
<td>1674</td>
</tr>
<tr>
<td><strong>Outcome</strong></td>
<td>0.50</td>
<td>0.24</td>
<td>0.33</td>
<td>1852</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>0.46</td>
<td>0.27</td>
<td><strong>0.34</strong></td>
<td>9246</td>
</tr>
</tbody>
</table>

One can assume that the insertion of POS tags as an input added more noise to the model instead of improving it, since F1-score is the lowest achieved so far compared with the previous two models that use word and character based information.

Figure 31: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values. Figure 31 below displays the accuracy values during training for the train and validation sets for every epoch.
As it can be seen in the diagram, accuracy on train set is improving across the epochs from 73.71% on the first epoch to a value of 84.74% on the last epoch. For validation set, accuracy curve follows a more flat course with a value of 77.91% on the first epoch and a value of 79.73% on the last epoch. It is worth mentioning that the validation accuracy curve during the training of this model does not have many fluctuations as the ones observed in Figure 29. But, in this model it is clear that performance is worse and final accuracy values on both datasets are lower than in the two previous models. Input data on this model, don’t help it to train and increase its performance significantly. On test set accuracy reaches the value of 79.1%.

![Accuracy results on train and validation dataset for every epoch during training of the neural network.](image)

*Figure 31: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values.*

The diagram on Figure 32 displays the loss metric curves during training on train and validation sets. The loss metric in both sets decreases. On the train set, loss value is 7.5550 on the first epoch, by assigning random values on the weights and in the final epoch loss value is 7.1994. In validation set, loss value on the first epoch is 7.3761, while in the last epoch is 7.3018. Finally, in test set loss metric is 7.2289.

From this diagram it is clear that the model achieved its best performance and cannot extract any new information from the data, since both loss curves are stabilized on the last epochs.
Figure 32: Loss metric for train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the loss values.

Table 4 indicates the evaluation results for every name entity for the BiLSTM – CRF model that uses an attention mechanism on POS embeddings and utilizes word, character and POS vector representations as an input to the neural network. It appears that the best F1-score is achieved again for the name entity Participant and reaches a value of 49%, while the lowest is achieved for the name entity Outcome with a value of 33%. The mean F1-score for this model is 36%.

Table 4: BiLSTM – CRF network with attention in POS embeddings.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.46</td>
<td>0.32</td>
<td>0.37</td>
<td>5057</td>
</tr>
<tr>
<td>Participant</td>
<td>0.53</td>
<td>0.46</td>
<td>0.49</td>
<td>663</td>
</tr>
<tr>
<td>Intervention</td>
<td>0.42</td>
<td>0.23</td>
<td>0.30</td>
<td>1674</td>
</tr>
<tr>
<td>Outcome</td>
<td>0.48</td>
<td>0.26</td>
<td>0.33</td>
<td>1852</td>
</tr>
<tr>
<td>Average</td>
<td>0.46</td>
<td>0.30</td>
<td>0.36</td>
<td>9246</td>
</tr>
</tbody>
</table>

It appears that the attention mechanism applied in POS embeddings improved classification results compared to the use of just POS embeddings. With the use of attention mechanism, POS tags that were more important obtained a higher score and the model was able to pay more attention to the significant vectors only. But even with the attention mechanism, the two previous models that don’t use POS tags information perform better than this model.
Figure 33 below displays the accuracy values during training for the train and validation sets for every epoch for the BiLSTM – CRF model with an attention mechanism on POS embeddings.

As it can be seen in the diagram, for the train set, accuracy is improving on every epoch. Finally it reaches an accuracy score of 85.10%. Although, accuracy on validation set does not follow the same course since it starts with a value of 77.10% and reaching its peak value on the 7th epoch with a value of 80.47% and then it is achieving a value of 78.99% on the last epoch. Regarding test data, accuracy is reaching a score of 78%.

The validation accuracy curve shows a behavior of the model, which is probably due to the dataset and is similar to the validation curve observed in the previous two models. The same fall of the curve someone can observe in the model that uses character based vector word representations and POS vector representations. It is very likely, that the model is using a random sample from the validation dataset and the validation set at each evaluation step is different and unknown to the model so far.

![Accuracy Results](image)

*Figure 33: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values.*

The diagram on Figure 34, displays the loss metric curves during training on train and validation sets. The loss metric in both datasets decreases, starting with loss on train set of 7.4994 and on validation set of 7.3743. In the last epoch during training, loss has decreased and reached a value of 7.1963 and 7.2988 on train and validation set respectively. Finally, in test set loss metric is 7.2267.
Figure 34: Loss metric for train and validation dataset for every epoch during training of the neural network. $X$–axis presents the number of epochs, while $y$–axis presents the loss values.

Table 5 indicates the evaluation results for every name entity for the BiLSTM – CRF model that uses an attention mechanism on the concatenated word and character embeddings and an attention mechanism on POS embeddings and utilizes word, character POS vector representations as an input to the neural network. It appears that the best F1-score is achieved once again for the name entity Participant and reaches a value of 53%, while the lowest is achieved for the name entity Outcome with a value of 27%. The mean F1-score for this model is 37%.

Table 5: BiLSTM – CRF network with attention in word character embeddings and attention on POS embeddings.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.44</td>
<td>0.32</td>
<td>0.37</td>
<td>5057</td>
</tr>
<tr>
<td>Participant</td>
<td>0.56</td>
<td>0.49</td>
<td>0.53</td>
<td>663</td>
</tr>
<tr>
<td>Intervention</td>
<td>0.48</td>
<td>0.32</td>
<td>0.39</td>
<td>1674</td>
</tr>
<tr>
<td>Outcome</td>
<td>0.47</td>
<td>0.19</td>
<td>0.27</td>
<td>1852</td>
</tr>
<tr>
<td>Average</td>
<td>0.46</td>
<td>0.31</td>
<td><strong>0.37</strong></td>
<td>9246</td>
</tr>
</tbody>
</table>

Figure 35 below displays the accuracy values during training for the train and validation dataset for every epoch.

As it can be seen in the diagram, during training accuracy on train set is increasing and it reaches its best value, 83.85% on the last epoch. On the validation set, the accuracy curve does not follow an upward course across all epochs, but after some fluctuations, it finally reaches its highest value on the final epoch, which is an accuracy score of 79.05%. Regarding test set, accuracy metric...
achieves a value of 80.2%. This number indicates that the model is not overfitting the data since the three accuracy scores do not have a big difference between them.

Figure 35: Accuracy results on train and validation dataset for every epoch during training of the neural network. X – axis presents the number of epochs, while y – axis presents the accuracy values.

Regarding loss metric across all epochs, Figure 36 shows the loss metric curve for train and validation sets. Both curves are decreasing, but as far as it concerns the train set, the fall of loss metric from the first epoch to the second and to the final epoch is more intense. On the first epoch, loss metric is 7.6292 for train data while in the last epoch it has dropped to 7.2136. Regarding validation dataset the loss metric curve is smoother. On the first epoch loss metric is 7.4318 while on the last epoch is 7.3263. On the test set, loss metric is 7.2450.
Finally, Table 6 indicates the evaluation results for every name entity for the BiLSTM – CRF model that uses an attention mechanism after the main BiLSTM layer and utilizes word, character and POS vector representations as an input to the neural network. It appears that the best F1-score is achieved once again for the name entity *Participant* and reaches a value of 50%, while the lowest is achieved for the name entity *Intervention* with a value of 38%. The mean F1-score for this model is 39%.

Table 6: BiLSTM – CRF with attention layer.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.47</td>
<td>0.34</td>
<td>0.40</td>
<td>5057</td>
</tr>
<tr>
<td>Participant</td>
<td>0.51</td>
<td>0.49</td>
<td>0.50</td>
<td>663</td>
</tr>
<tr>
<td>Intervention</td>
<td>0.49</td>
<td>0.32</td>
<td>0.38</td>
<td>1674</td>
</tr>
<tr>
<td>Outcome</td>
<td>0.49</td>
<td>0.26</td>
<td>0.43</td>
<td>1852</td>
</tr>
<tr>
<td>Average</td>
<td>0.48</td>
<td>0.33</td>
<td><strong>0.39</strong></td>
<td>9246</td>
</tr>
</tbody>
</table>

Figure 37 below displays the accuracy values during training for the train and validation set for every epoch for the last model that uses an attention layer after the main BiLSTM layer.

Like in the rest of the models, on the train set, accuracy curve follows an upward course, starting from a value of 72.52% and achieving a final value of 84.10%. For the validation set, once again the accuracy curve has a big fall on the second epoch and after that it increases to a value of 80.07%. This fall on the second epoch occurs because the first epoch wasn’t enough to train the
model properly. Another reason could be the fact that the model is using a random sample from the validation dataset and the validation set at the second evaluation step is different and unknown to the model so far. The course on both curves, indicates that there is no overfit of the model on the train data. On test set, accuracy achieves a score of 80.2%.

![Figure 37: Accuracy results on train and validation dataset for every epoch during training of the neural network. X - axis presents the number of epochs, while y - axis presents the accuracy values.](image)

Figure 38 displays the loss metric values for every epoch during training on train and validation set. Both curves are decreasing, but as far as it concerns the train set, the fall of loss metric from the first epoch to the second and to the final epoch is more intense. On the first epoch, loss metric is 7.6572 for train data while in the last epoch it has dropped to 7.2167. Regarding validation set the loss metric curve is smoother and it is clear that it stabilizes on the last epochs of training. On the first epoch loss metric is 7.4227 while on the last epoch is 7.3096. On the test set, loss metric is 7.2325.
Figure 38: Loss metric for train and validation dataset for every epoch during training of the neural network. X-axis presents the number of epochs, while y-axis presents the loss values.

On Table 7 below, there are the precision, recall and F1-score values for every entity in every different architecture implemented on this thesis. \textit{W-BiLSTM-CRF} refers to the BiLSTM – CRF model that uses word vector representations as input. \textit{WC-BiLSTM-CRF} refers to the BiLSTM – CRF model that uses word and character vector representations as input. \textit{WCP-BiLSTM-CRF} refers to the BiLSTM – CRF model that uses word, character and POS vector representations as input. \textit{WC-AP-BiLSTM-CRF} refers to the BiLSTM – CRF model that uses an attention layer on the POS embeddings. \textit{AWC-AP-BiLSTM-CRF} refers to the BiLSTM model that uses attention layer on word, character embeddings and on POS embeddings and finally, \textit{WCP-BiLSTM-A-CRF} refers to the BiLSTM – CRF model that uses an attention layer on the output of the BiLSTM layer.

Table 7: Table of aggregated results for every name entity of the different neural network models implemented.

<table>
<thead>
<tr>
<th>Entities</th>
<th>Participant</th>
<th>Intervention</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>F1-score</td>
</tr>
<tr>
<td>\textit{W-BiLSTM-CRF}</td>
<td>0.54</td>
<td>0.48</td>
<td>0.51</td>
</tr>
<tr>
<td>\textit{WC-BiLSTM-CRF}</td>
<td>\textbf{0.58}</td>
<td>\textbf{0.50}</td>
<td>\textbf{0.54}</td>
</tr>
<tr>
<td>\textit{WCP-BiLSTM-CRF}</td>
<td>0.51</td>
<td>0.50</td>
<td>0.50</td>
</tr>
</tbody>
</table>
Table 8 presents the aggregated results of mean values for precision, recall and F1-score for every model.

**Table 8: Table of aggregated average results for every name entity of the different neural network models implemented.**

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>W-BiLSTM-CRF</td>
<td>0.45</td>
<td>0.32</td>
<td>0.38</td>
</tr>
<tr>
<td>WC-BiLSTM-CRF</td>
<td>0.48</td>
<td>0.32</td>
<td>0.38</td>
</tr>
<tr>
<td>WCP-BiLSTM-CRF</td>
<td>0.46</td>
<td>0.27</td>
<td>0.34</td>
</tr>
<tr>
<td>WC-AP-BiLSTM-CRF</td>
<td>0.46</td>
<td>0.30</td>
<td>0.36</td>
</tr>
<tr>
<td>AWC-AP-BiLSTM-CRF</td>
<td>0.46</td>
<td>0.31</td>
<td>0.37</td>
</tr>
<tr>
<td>WCP-BiLSTM-A-CRF</td>
<td><strong>0.48</strong></td>
<td><strong>0.33</strong></td>
<td><strong>0.39</strong></td>
</tr>
</tbody>
</table>

Based on the reported results indicated on the table above, an overall best performance is achieved for Model 6, but this model does not perform the best regarding every entity.

Model 6, that uses word, character and POS vector representations as input and an attention layer after the main BiLSTM layer has the best overall performance with F1-score reaching the value of 39%, a precision score of 48% and finally, a recall score of 33%. These results are the best among the other average results of the rest of the models. This architecture has the best precision, recall and F1-score compared to the values presented for the other models.

Although this model appears to have the best overall performance, it does not perform equally well for all the name entities. From Table 7 it seems that it has the best performance only for the name entity *Outcome*. Regarding the name entity *Participant*, all the models have their best performance for this tag. Although all of them achieve their highest score for this entity, *Model 2* appears to have the best performance with an F1-score of 54%, a recall score of 50% and a precision score of 58%. For the name entity *Intervention*, the best score on all metrics is achieved by *Model 5*. *Model 5* predicts this tag with an F1-score of 39% while recall and precision values are 32% and 48% respectively.

These results indicate that there is not one model suitable for all name entities in this dataset. In addition, a model with less information as input may be suitable and perform well in predicting an entity, i.e *Model 2* with name entity *Participant*. This model uses word and character level information as input and performs better than the models that also use POS information. But for other name entities, i.e *Outcome*, additional information are needed as input for better prediction,
such as POS information and the attention mechanism. This observation can be referred to the fact that some entities, like *Participants*, are more distinct on the dataset, while others, like *Intervention* are vaguer. A word that is tagged as an Intervention in one sentence could be tagged as an Outcome or has no tag in another sentence.
Chapter 6 Conclusions

This thesis presents six neural architectures for named entity recognition from medical publications. Different inputs and different layer structures were tested on the neural network models in order to discover the most efficient architecture in terms of performance. POS tag information were compared with more traditional sources, such as word and character – based word representations to examine if it would improve a model’s performance or not. New layers were also tested, such as attention layer, which is a very promising field in the task of machine translation and can be explored in other fields as well, such as sequence tagging.

A key aspect of the models, is that they model output label dependencies via a simple CRF architecture. Word representations are crucially important for success, either through pre – trained word representations (GloVe), or through character – based representations that capture morphological and orthographical information. In addition, word representations through POS tag representations that capture grammatical information can improve the performance of the model if they are used with an attention layer.

The analysis conducted in the context of this thesis indicates that this approach does not work well because of the difficulty to arrive at a consistent tagging of the PICO elements. Although, results are very promising and could be improved. Model improvement can be achieved with various methods. One solution could be relevant to feature extraction techniques from the dataset. Currently, the neural network models used in this implementation do not use any extra features apart from vector representations of sentences. Feature extraction could add extra information to these models and improve their performance. Another significant factor that could change and further improve the results is the hyper parameter optimization of the neural network models. Unfortunately, tuning a neural network requires resources that were not available during the implementation of this thesis. But it is possible that the implemented models would perform differently with a different optimizer or activation function or even different number of nodes in every layer.

Although, the most crucial role in the performance of a prediction system is the dataset. Datasets play a key role in sequence tagging problems, especially in EBM problems were a lot of entities are mixed and can be tagged as different entities depending on the sentence they are part of. The specific dataset, used in this thesis, EBM – NLP, is relatively new to the research community and haven’t been used in a significant number of implementations. Hence, there are not a lot of conclusions to extract regarding its efficacy on sequence tagging problems.
References


