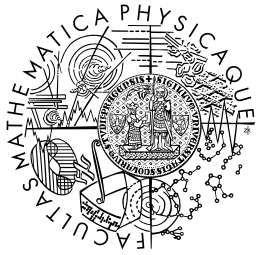

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Shadasha Williams

Named Entity Recognition in the Biomedical Domain

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: prof. RNDr. Pavel Pecina, Dr.
Study programme: Computer Science
Study branch: Mathematical Linguistics

Prague 2021

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague ..... date 21/7/2021 ... Shadasha Williams .....

Author's signature

The work of thesis is dedicated to all of my professors as well as my peers I have met at Charles University, mainly those in the UFAL department. My experience has been enriching and the faculty has provided me with the knowledge and tools that were necessary to carry out the work on this thesis.

Title: Named Entity Recognition in the Biomedical Domain

Author: Shadasha Williams

Department: Institute of Formal and Applied Linguistics

Supervisor: prof. RNDr. Pavel Pecina, Dr., Institute of Formal and Applied Linguistics

Abstract: Information extraction is an integral piece to understanding data, mainly we attempt to extract what is meaningful in the data. In regards to linguistic data one sub-task of information extraction that we can focus on is named entity recognition. For the purpose of this thesis we will focus on the biomedical domain. The goal of this thesis is to get an understanding of the relevant and promising methods for named entity recognition, mainly state of the art results using neural networks and deep learning approaches.

Keywords: Named Entity Recognition, Deep Learning, Neural Networks, Word Embeddings, Biomedical Named Entity Recognition

Contents

Introduction **2**

1 Thesis Introduction **3**
1.1 Thesis Motivation . . . 3
1.2 Linguistic Background . . . 3
1.3 Natural Language Processing . . . 4
1.3.1 Biomedical Natural Language Processing . . . 5

2 Named Entity Recognition **8**
2.1 Named Entity Recognition Task . . . 8
2.2 Named Entity Recognition Methods . . . 8
2.2.1 Dictionary Based Approaches . . . 8
2.2.2 Rule Based Approaches . . . 9
2.2.3 Machine Learning Approaches . . . 10
2.2.4 Named Entity Recognition Evaluation Measures . . . 16
2.3 Issues in Named Entity Recognition . . . 17
2.3.1 Ambiguity . . . 17

3 Word Embeddings **22**
3.1 Sub-word Embeddings . . . 26
3.2 Contextualized Embeddings . . . 29

4 Methodology **36**
4.1 Modules and Tools . . . 36
4.2 Sources . . . 38
4.2.1 PubMed . . . 38
4.2.2 NCBI Disease Corpus . . . 39
4.3 Ontologies . . . 39
4.4 Experiments . . . 41
4.4.1 Performance on Gold Standard . . . 47
4.5 Conclusions . . . 49
4.5.1 Future and Related Work . . . 51

Bibliography **52**

List of Figures **57**

List of Tables **58**

List of Abbreviations **59**

A Attachments **60**
A.1 First Attachment . . . 60

Introduction

Language is one of the phenomena of the human experience, our unique and complex communication modes continue to distinguish us from our animal counterparts. Human communication has expanded from a collection of tones and sounds to prose, poetry, and studies. In the recent decades there have been numerous studies addressing the origins and evolution of language. Some scientists theorize that human language is a result of how human's brains have evolved over time, proposing that the development of parietal, occipital, and temporal lobe provided humans with the tools for "conceptual structure" (Wilkins and Wakefield [1995]). Other studies link the evolution of human language to the unique stages of life, in other words ontogeny (Locke and Bogin [2006]). The work of Locke and Bogin suppose that the unique patterns of the human life cycle namely childhood and adolescence contribute to the evolution of human language (Locke and Bogin [2006]). As we continue to explore the origin of language we too leave a trail for future understanding of human behavior and knowledge. Artifacts from language have expanded exponentially with the new age of data, documenting many of the processes which allow our society to continue to function. This richness of data has drawn me not only to linguistics but to the processing of linguistic data particularly for the life sciences. In the following chapter I will discuss my motivation for this thesis and outline the tools and concepts that were necessary for me to complete this body of work.

1. Thesis Introduction

1.1 Thesis Motivation

The search for knowledge for better understanding is essential for progression. This philosophical principle can be applied to any discipline and is indisputable. In the discipline of information extraction and Natural Language Processing, insight can be gained through a multitude of mechanisms. In my own work, there has been a need for gaining insight into unstructured data. As an engineer and modeler of practices for information extraction, I am fully informed in the usage of information extraction in the biomedical/pharmaceutical domain. Some of the use cases in the highly regulated and exploratory industry involve detection of documents necessary for anonymization due to the sensitive data contained, exploring new chemical or biological structures, and general information extracting, to name a few. One of the tasks necessary for the aforementioned processes is a Named Entity Recognition, namely a domain specific tool. Biomedical markers and correlations are critical for growth in the field for the understanding of bio-processes and interactions between biomedical entities, such as diseases and drugs or genes and diseases. There is currently a plethora of information online in the scientific and biomedical domain. In this thesis I will be focusing on the database resources from The National Center for Biotechnology Information (NCBI) (Wheeler et al. [2007]). The NCBI has a number of tools and resources for biomedical information, of which one reliable source that researchers utilize is PubMed Central, as it is a readily available archive with a focus on journals of biomedical and life sciences. In order to compensate for the generality of the current state of the art named entity recognizers and not rely on the static ontologies sourced from gazetteers, a bio-medically trained entity recognizer is proposed using PubMed data for training and a deep learning approach using LSTM variations is examined (Hochreiter and Schmidhuber [1997b]). The goal of this thesis is to:

- examine the current neural network approaches to named entity recognition of the biomedical domain using recurrent networks
- explore the influences of word embeddings in regards to the NER task
- investigate the effects of word embeddings on ambiguous text.

1.2 Linguistic Background

Language is a phenomenon of human communication and interaction. It allows for humans to distinguish themselves amongst other species and is incredibly diverse in its form and structure. Linguistics is the study of language, its components, and how humans communicate. Linguistics attempts to analyze the form, meaning, and contextual importance of language. In order to understand this thesis, some key ideas in linguistics must be defined. Context, an abstract concept for information sharing, is a frame that captures a focal event and equips the listeners with the appropriate resources for the understanding and interpretation

of the shared information. Focal events are a critical part of encoding information in human interaction and are the highlighted events in the exchange. Thus focal events (focus) and context share a complementary relationship (Duranti and Goodwin [1992], Krifka [2008]). Focus and contexts are critical sub-units in information structure, a component of language in which speakers package information to their interlocutors. Language cannot merely be broken into sub-units and in order to understand its complexity we must also recognize the levels in which language functions. Linguists have described the levels of language as language stratification (Halliday and Matthiessen [2013]). Apart from context

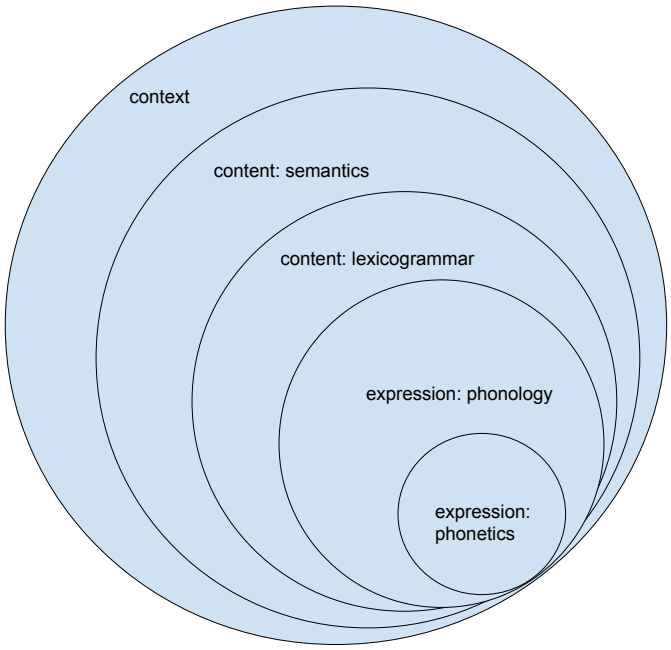

Figure 1.1: Language stratification, adapted from Halliday and Matthiessen [2013]

Halliday and Matthiessen expand the content of language as semantics and lexicogrammar (Halliday and Matthiessen [2013]). The lexicogrammar includes the grammar and vocabulary, and the semantics carries the meaning of the content. The strata or 'levels' that provide us with the ability to make such expressions are phonology the manner in which we organize sounds into structure and phonetics, the physical properties necessary for human speech (Halliday and Matthiessen [2013]). These stratal planes will be useful for understanding the ways that we approach language from a non human perspective.

1.3 Natural Language Processing

Natural language processing (NLP) as defined by Jurafsky and Martin is the interdisciplinary field which aims to use computers to perform tasks which involve human language (Jurafsky and Martin [2008]). As human interaction is complex, some the tasks related to natural language understanding and processing have

proven to be equally complex. Some tasks in NLP include speech generation, machine translation, and information extraction. There are several approaches to solving NLP tasks, they can be grouped into four categories: symbolic, statistical, connectionist, or combination of aforementioned categories (Liddy [2001]). The symbolic approach previously dominated natural language processing approaches, it uses information about the language to perform deep analysis of the language (Dale [2000]). Some examples of symbolic approaches include rule based systems and semantic networks and its techniques include decision trees, K nearest neighbor algorithms and conceptual clustering (Liddy [2001]). Statistical approaches to NLP don't require facts about the language and instead use mathematical techniques to model large data sets of language called *corpora*. These corpora contain many samples of real world text and are used a basis for models to generalize the language patterns. More information about the techniques used in statistical NLP will be discussed in later chapters. The last distinct approach to NLP is the connectionist approach, which much like the statistical approach builds models based on generalizations from large data sets, the main difference between the two is that connectionist models also use different theories of representation (Liddy [2001]). One of the most common connectionist approaches consist of a network of computing units. These units contain input and outputs and can be connected to other units in the model with weights, each unit is also assigned a numerical value called the activation level (Selman [1989]). More information about the connectionist approach will also be discussed in further chapters. The last approach combines the previous approaches to create a hybrid approach to natural language processing. Each of the approaches can come with its drawbacks so combining different models can create leverage.

1.3.1 Biomedical Natural Language Processing

Biomedical natural language processing or BioNLP combines the NLP approaches to the biomedical field to help aid some of the processes that researchers and clinicians currently face. Using NLP techniques for solving biomedical issues isn't a new phenomena, the interdisciplinary field of bio-informatics has been introducing methods from NLP during conferences and in scientific journals (Blaschke et al. [2002]). The idea that we could use information retrieval to gain insights from existing public was proposed by Don Swanson, his ideas about "undiscovered public knowledge" helped create the model which BioNLP still uses today, which supposes that within our current knowledge bases there exists "new knowledge" which has yet to be discovered (Swanson [1986b]). Swanson used the principle of "new knowledge" to drive the development of hypothesis for treatments for Reynaud's syndrome as well as create hypothesis for links between migraine and magnesium (Swanson [1986a], Swanson [1988]). Swanson's defined the paradigm called the ABC model, which investigates the idea that information A is linked to information B in one source and another may link B to information C, if the connection between A and C has not yet been reported we can regard the "hidden" connection as new knowledge, which can be further explored Blaschke et al. [2002]. Mining scientific literature has become one of the most common use cases in BioNLP, text mining can provide a method for making this process more automated. Text mining uses several NLP techniques to extract information:

information retrieval, information extraction, and data mining (Ananiadou et al. [2006]). The three techniques mentioned above allow for the collection of relevant information, the extraction of specific information, and the development of connections and associations within texts (Ananiadou et al. [2006]). Hypothesis generation, a task which was Swanson modeled using text mining and eventually used successfully to generate hypothesis, can use text mining in the framework mentioned by Ananiadou et al. Hypothesis generation is just one of tools which gives researchers in the biomedical field a helping hand, it can allow for quicker discovery and can redirect the time spent manually connecting information from large scientific repositories. Knowledge discovery using text mining isn't the only NLP process being used in the biomedical domain, QA (question answering systems), text summarization, machine translation for translating documents into a target language, and even sentiment analysis (Friedman and Elhadad [2014], Zucco et al. [2017]). While much of the work in BioNLP is using the algorithms on data within the biomedical domain, another large effort in the BioNLP community is the generation of annotated corpus. Several corpora have served as source for further experimentation in BioNLP, as many of the algorithms require training data which needs to be annotated (Kim et al. [2003], Pyysalo et al. [2007]). Corpora like the GENIA and others allow for there to be a standard within BioNLP and named and are continually used and expanded (Makino et al. [2002], Cimiano et al. [2006]).

2. Named Entity Recognition

This chapter is an introduction to the key methods and concepts of this thesis. It defines the current issues and solutions in named entity recognition. This chapter has been divided into several sections and will describe the tasks, methods, and evaluation of named entity recognition.

2.1 Named Entity Recognition Task

Information extraction is the sub-field of natural language processing that deals with the automatic retrieval of information from unstructured data. Currently, due to the digital age, there is a collection of large linguistic data, a critical resource for both academics and businesses using NLP. While the volume of data to be analyzed is incredibly rich, often, the digital traces of human interactions and records are unstructured and difficult to analyze. In order to identify key information in the larger scope, researchers in information extraction identified key information units, which they coined a "named entity" (Nadeau and Sekine [2007a]). Sample sentences with the named entities highlighted are displayed below:
Bill Gates *of* ***Microsoft*** *is said to be one of the greatest minds of our time.*
The collaboration of ***the Association of Computation Linguistics*** *and industry has been tremendous at this conference.*
These key units are not limited to names of people but also include organizations, concepts, dates, locations, and can also include numerical values such as unique identifiers, and monetary values. Named entity recognition is a sub-task in information extraction that recognizes named entities. This task aims to locate and categorize the entities of interest (Mohit [2014]). In the above sample, recognizing the entity can be seen as providing the index of the named entity, and categorizing it would be providing the additional label of person, for *Bill Gates*.

2.2 Named Entity Recognition Methods

There are several approaches to named entity recognition, mainly this section will discuss the different methods which have been used to the present day state of the art approaches.

2.2.1 Dictionary Based Approaches

Dictionary based approaches for named entity recognition are the most simplistic of the approaches. They are based on predefined lists of entities which are often referred to dictionaries or gazetteers. The gazetteers are then used as the source for string matching, which simply detects the terms, returning matches in the text. Depending on the algorithm used, terms can be matched exactly or partially using fuzzy matching. One string matching method that can be used is the Aho-Corasick algorithm (Aho and Corasick [1975]). The Aho-Corasick algorithm builds a finite state machine from the words in the dictionary. The

input to the algorithm is the text, and the output is locations where keywords occur within the text, if they exist. Algorithms such as the Aho-Corasick have proven to be fast and efficient ways for string matching and can be run in $O(n + m + z)$ time, where n, m, z represents the length of the input string, the total number of characters in each word, and the total number of occurrences of words in each text, respectively. Another approach to dictionary based NER uses edit distance to measure similarities between possible entities in the text to entries in the dictionary, to account for misspellings and possible matches that are not present in existing dictionaries (Wang et al. [2009]). Within the biomedical domain dictionary approaches have shown promising results and have been used for protein and gene recognition (Egorov et al. [2004], Hirschman et al. [2002]). Dictionary based methods still have a few requirements in order to be a reliable model. Data sources which dictionaries are built on must be reputable and there is still typically a pipeline for processing the dictionary to filter it from possible ambiguities and noise, like common words and short dictionary entry removal. The text is also typically pre-processed before being "matched" against the dictionary. This pre-processing can include tokenization, lemmatization, and normalizing the casing of text. There are many reputable knowledge sources in the biomedical domain, such as the Open Biomedical Ontologies (OBO) Foundry and the Unified Medical Language System (UMLS) which include a large number of biomedical vocabularies (Bodenreider [2004], Smith et al. [2007]). These knowledge sources have become standards in the biomedical domain, but this approach even with the best source dictionaries can be limited. One of the main issues with the dictionary based approach is that it limits the entities only to pre-existing entities, new unseen entities are not accounted for. The dictionary approach can have good results but in order to create a promising system that is dynamic other methods should also be employed.

2.2.2 Rule Based Approaches

One of the first approaches to named entity recognition was done by the use of hand crafted rules and heuristics, this approach is generally regarded as the rule base approach. A good example of this approach was presented by Lisa Rau (1991), who used a series of rules based off of text casing, patterns that are associated with the entity (in the case of company detection the presence of suffixes such as Inc., Corp.), and some contextual heuristics regarding the surrounding words (Rau [1991]). Another example of a rule based approach is the FASTUS framework. FASTUS is a system that consists of four main steps to recognize entities: triggering, recognizing phrases, recognizing patterns, and merging incidents (Appelt et al. [1993]). First, for the given sentence input specific words are identified as "trigger" words, which are predefined for each entity. In the next steps, critical word-class clusters are identified, for example, noun phrases, verb phrases, and are input for pattern recognition. Relevant phrases are then transformed into finite state machines, where the transition states are represented as the grammatical phrases identified in the previous step. The final step of the FASTUS process is merging and formatting the incidents to be unified of the same type. Features of the text, include linguistic features, like the casing, or presence of prefixes of suffixes in the words (Nadeau and Sekine [2007a]).

Features can also include information about the entire document set, such as the corpus frequency and the location of the entities in the entire document (Nadeau and Sekine [2007a]). Features allow rule based models to make decisions beyond simple pattern detection, and for easier understanding on how models operate.

2.2.3 Machine Learning Approaches

Machine learning methods attempt to solve some of the limitations of the rules based model, as they attempt to *learn* the features of a specific entity of interest. Machine learning methods frame the problem with a statistical model for predicting if a word in a sequence matches with an entity tag. There are two main approaches to named entity recognition in regards to ML, the supervised and unsupervised approach. In the supervised approach, we provide a model with annotated training data, methods to solve the issue, and test the methods on data that have not been seen by the model (test data). Unsupervised approaches, on the other hand, do not rely on annotated data and attempt to explore the underlying structure of the input data. Named entity recognition can then redefined in the lens of machine learning as a classification algorithm, classifying if a sub-unit is a member of an entity group or not.

Hidden Markov Models

One of the first transformations of NER into the machine learning domain is based on the logic of Markov processes (Bikel et al. [1999]). It follows an HMM model, which structurally consists of states and observations. The entities of interest in an HMM NER model are represented as states, and the observations are the sequence of words in a sentence (Bikel et al. [1999], Baum and Petrie [1966]). An HMM consists of:

- a set of states of size N; $Q = q_1 q_2 ... q_N$
- a matrix consisting of transition probabilities; $A = a_{11} ... a_{ij} ... a_{NN}$, each representing the probabilities of moving from state i to state j
- a sequence of observations; $O = o_1 o_2 ... o_T$ of T total observations, from a vocabulary defined as $V = v_1 v_2 ... v_V$.
- a set of emission probabilities, expressing the likelihood of a particular observation o_t from a particular state; $B = b_i(o_t)$
- initial probability distributions of states $\pi = \pi_1, \pi_2, ..., \pi_N$

(Jurafsky and Martin [2008]).

The figure below shows the connections of the states of model HMM for NER. Special states are introduced for sentence boundaries that have connections to each entity state. Entity states also are reflexive and are fully connected to all other entity states in the graph. The model mentioned in the paper is slightly modified to contain an additional state for non-entities. The HMM model is generative, such that it can create the sentence and the tags using Bayesian principles.

$$P(\boldsymbol{E}|\, \boldsymbol{W}) = \frac{P(\boldsymbol{W}, \boldsymbol{E})}{P(\boldsymbol{W})}$$

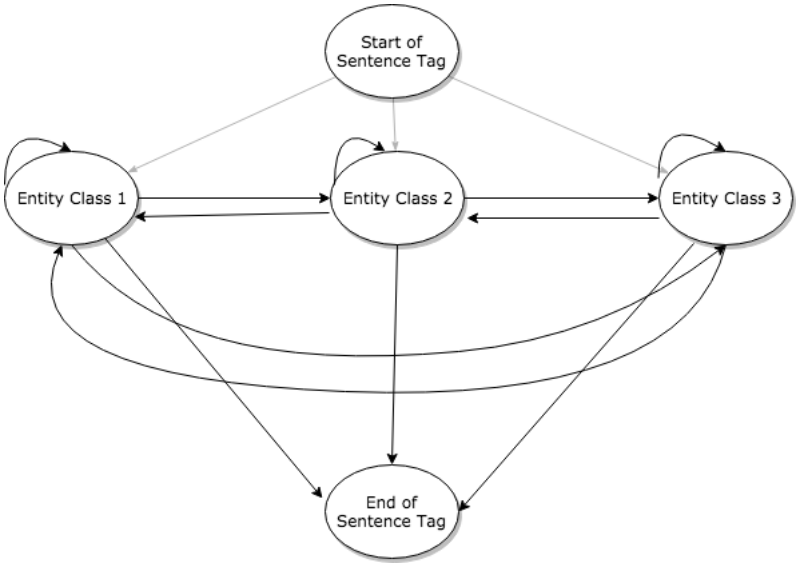

Figure 2.1: Sample structure for HMM NER Model, adapted from (Bikel et al. [1999]), Figure 3

Given a sequence of words $\boldsymbol{W}$ we can predict a sequence of entities $\boldsymbol{E}$, using the Viterbi algorithm we can iterate through the name classes to find

$$Max(P(E|W))$$

(Viterbi [1967]). There is an additional layer of features in the model, which gives us additional information about the word. In the model described, words are transformed into a tuple of *(word, feature)*. An example of some of the features is word classes, special characters, and case descriptions. The features described are language-specific but are only a small piece of the entire algorithm, so the framework remains dynamic for other language uses. Training the model is as follows, using a given sample, we get the probabilities based on the counts of entities and words seen in the training data. We then find the probability of the first word given the counts of the entity class and previous word counts. Due to the nature of training data, some changes need to be made for the lack of seen entities and words in training. For this purpose, they introduce back-off models and smoothing into the model to correct the probabilities for unknown input. While HMM models have shown favorable results, some of the drawbacks include the performance dependency on the input data. If the language model is similar to the one used in the paper, a bi-gram model, then it has a limited window of information for entities. There is also the issue mentioned in the paper of hierarchical entities, i.e., *Bank of America*, while it is a stand-alone entity, the model described above only captures sub-units as entities in the case *America* would be the only entity extracted.

Conditional Random Fields

The HMM model discussed in the previous section is an example of a *generative* model. This section will explore a *discriminative* approach to NER. The key differences between the two are the approaches to such; generative methods attempt to characterize the entities of interest by learning the distribution of the

entire data set. Discriminative models focus on the boundaries between positive and negative examples; this approach is limited to supervised machine learning methods as the labeled data is key for the learning of that boundary. Conditional Random Fields (CRFs) are an example of a discriminative approach to NER (Sutton et al. [2012]). The input for a CRF, like HMM, is a sequence of data, in our case, a sentence, and the output is a sequence of entities. Unlike HMM models, the CRF is not limited to the current word and previous tag. Instead, CRFs explore more features of the input using feature vectors . Each word in the sequence can be represented as

$$x_i = \{x_1, x_2, \ldots, x_T\}$$

A sample of some of the features that can be extracted for our task is the current or previous word or words within a given window, orthographic features, and prefixes/suffixes of the word. An important concept for understanding the notions of CRFs is linear-chain CRFs. This distribution, given the input vectors x and y, parameter weights $\theta = \{\theta_k\} \in \Re^K$, and feature functions $\mathbf{F} = \{f_k(y, y', x_t)\}_{k=1}^K$, we can formalize the chain as:

$$P(\mathbf{y}|\mathbf{z}) = \frac{1}{Z(x)} \prod_{t=1}^{T} exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t)\}$$

where $Z(\boldsymbol{x})$ is an input-dependent normalization function:

$$Z(x) = \sum_{y} \prod_{t=1}^{T} exp\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t)\}$$

(Sutton et al. [2012]). CRFs have the ability to slightly more dynamic than the aforementioned HMM models as the transition states remain static regardless of input. In the CRF model, we can create a dependency between the transition score and the current input vector features. For our task, this is especially useful as entity recognition is highly contextual in nature, and the current observation vector allows for a more specialized model. Many of the state-of-the-art NER tools use CRFs because there are no assumptions from the model that the features are independent and the model remains dynamic in the labeling process.

Neural Network Approaches

Neural networks are networks comprised of connected layers, which use activation functions, processors, and weights to build estimates. Neural networks have been a popular topic in machine learning, surpassing state-of-the-art performance in many natural language processing tasks. This section will discuss the methods and architectures of neural networks used for NER.

Long Short-Term Memory networks (LSTMs) have shown remarkable results in named entity recognition (Hochreiter and Schmidhuber [1997a]). LSTMs are a type of recurrent neural network (RNN), networks used for sequential data (Pascanu et al. [2013]). RNNs operate on a sequence of vectors and, in turn, output a sequence of vectors representative of information about the input data at each step in the sequence. As information is propagated through a recurrent neural network, previous time points are used as inputs to the next layer. At

each layer, there is a cost function designed to weigh the error of the predicted outcomes. In the simplest RNN architecture, each state h_t is defined as $h_t = f(h_{t-1}, x_t)$. The architecture for the network can be seen in Figure 2.2 below. The output of the network is a prediction vector, which will be used in conjunction with a function to propagate through the network and update the weights to minimize the error. The backpropagation used in RNN, is called back-propagation through time, (BPTT) it is a gradient based technique that traverses the network through time using the inputs, outputs, and a set of parameters. The following variables are defined for the original backpropagation algorithm:

- x: *the input vector*
- $\hat{y}$: *the predicted target vector*
- y: *the target vector*
- L: *loss function*
- $w_a ba$: parameters used to compute the activation at each time step
- a^t: the activation function at a time step t
- $w_y b_y$ parameters used to compute $\hat{y}$

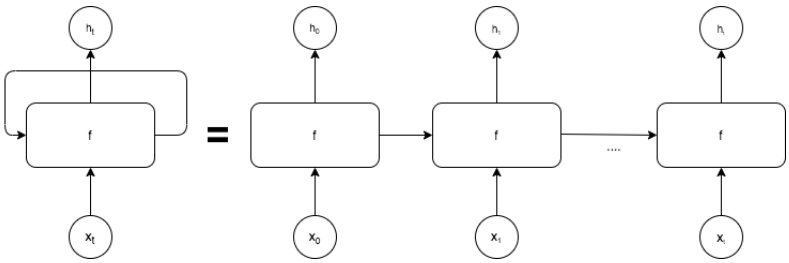

Figure 2.2: Architecture of recurrent neural network, adapted from Olah [2015]

Typically we will use cross-entropy loss $L^t(\hat{y}^t, y^t) = -y^t \log \hat{y}^t$ at a given time t, thus the overall loss for the entire sequence as $L(\hat{y}, y) = \sum_{t=1}^{T} L^t(\hat{y}^t, y^t)$. Using these inputs, we can compute losses for each step and compute their sums going right to the left or "backward" in time. Using the backpropagation algorithm but starting from the error, we calculate the gradients by chain rule of differentiation and sum up these gradients. The issue that we run into with traditional RNNs is that with each stage, we are multiplying the inputs, and multiple passes lead to the derivatives exploding or vanishing. While exploding gradients can be solved by clipping the gradients with a threshold, the vanishing gradient presents a more difficult issue. LSTMs provide a solution for the vanishing gradient problem, and this is done by the architecture of the LSTM gradient itself. In figure 2.3, details of how the cell retains its memory are shown.

In the figure above we still have the input x_t and the output h_t, and we also have C_{t-1} as well as C_t. Together these components represent the namesake of the network and corresponds to both long and short terms. The cell contains four gate functions, the forget gate F_t, the input date I_t, the candidate memory C_t, and the output gate O_t. The σ represents a fully connected layer with activation

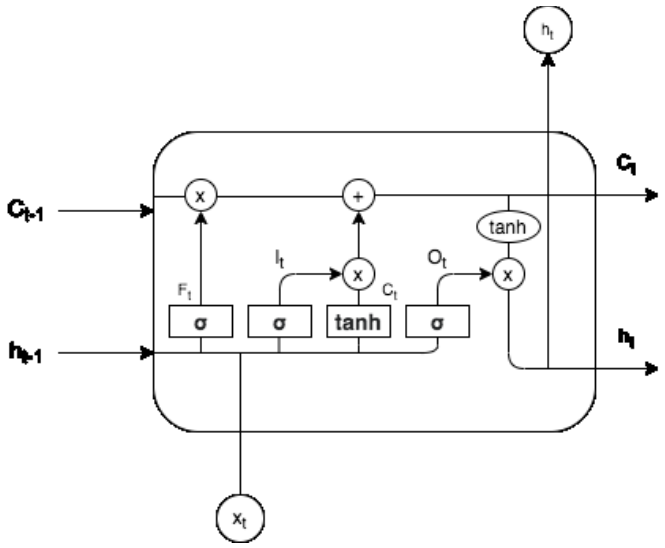

Figure 2.3: LSTM Cell, adapted from Olah [2015]

function, and the tanh represents a tanh function whose values range from [-1,1]. The input, forget, and output are calculated in the same manner:

$$I_t = \sigma(x_t w_{xi} + h_{t-1} w_{hi} + b_i)$$

In the input gate calculation above w_{xi} and w_{hi} represent weight parameters for the input and b_i as the bias parameters for the input I. Each of the gates have their own weight and bias parameters thus w_{xf}, w_{hf}, b_f, and w_{xo}, w_{ho}, b_o represent the weights and bias parameters for the forget and output gates respectively. The memory cell is a critical feature of LSTMs which function like a hidden state, but are designed to remember additional information. Before being explicitly defined a candidate memory cell is first constructed. Its calculation differs slightly from the aforementioned gates, and follows the equation:

$$\hat{C} = \tanh\left(x_t W_{xc} + h_{t-1} w_{hc} + b_c\right)$$

, the function of $\hat{C}$ is to governs how much new data should influence the process. Using the input, candidate memory and forget a new update equation is used $\hat{C} = F_t \odot C_{t-1} + I_t \odot \hat{C}_t$, where the $\odot$ represents element wise product. Hidden states are then computed as $H_t = O_t \odot \tanh C_t$.

Understanding how the LSTM cell operates is critical for understanding the currently used approaches to neural NER, as it will further build on the LSTM structures. In the paper *Neural Architectures for Named Entity Recognition*, Lample et al. combine both CRFs and a bidirectional LSTMs for a model for NER, which is the approach that will also be used in this thesis work [Lample et al., 2016]. The input sentence X is represented as $(x_1, x_2, ..., x_n)$, where each word x_i is a vector with a predefined dimension. This input is then propagated to an LSTM that calculates a representation of the context on the left of a given the word x_i in the sequence. In order to gain insight into both the left and the right context, another LSTM reads the input sequence in reverse to compute the right context. The two left, and right representations are then concatenated to create h_t. In the previous section, CRFs usage is discussed as they are often the preferred

statistical method for NER as they do not predict independently. Rather they take into account the neighboring samples. The model's final layer is then a CRF tagging model that uses the input from the word representations. This is done by using the information generated by the Bi-LSTMs, a matrix of n (*number of words in the sentence*) by k the number of possible tags, the matrix can be read as scoring of the word and tag combination. The prediction sequence Y, where $Y = (y_1, y_2, ..y_n)$, is scored with its input X using the following equation:

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

where $P_{i,j}$ maps to the score of the j^{th} tag of the i^{th} word, and $A_{y_i, y_{i+1}}$ is the transition score from the tag i to j. $P_{i,j}$ are calculated by taking the dot product of the word embedding and a previously defined tagging model scores, and then finally are combined with the bi-gram probability scores (the models built by Lample et al. only model bi-gram interactions). To calculate the probability of the sequence Y a softmax over all of the possible tag sequences using the equation:

$$p(Y|X) = \frac{e^{s(X,Y)}}{\sum_{\hat{y} \in Y_X} e^{s(X, \hat{Y})}},$$

where Y_X is all of the possible tag combinations that X can represent. The log-probability is maximized during training, using the equation

$$\log(p(Y|X)) = s(X, Y) - log(\sum_{\hat{y} \in Y_X} e^{s(X, \hat{Y})})$$

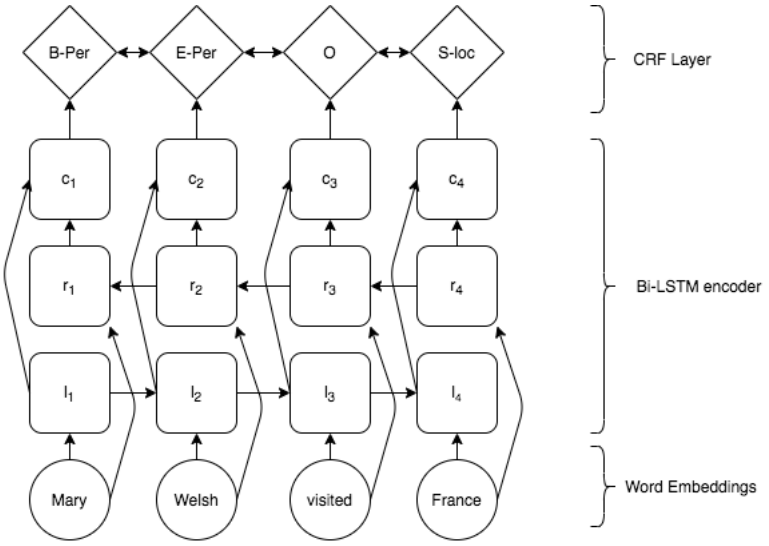

Figure 2.4: Architecture of the network, adapted from Lample et al. [2016], where l_i and r_i represent left and right contexts, and the contextual representation c_i is produced from the concatenation. This figure used the IOB tagging schema.

During the decoding, the prediction of the output sequence depends is chosen by the sequence that has the maximum score $y*$, where:

$$y* = argmax_{\hat{Y} \in Y_X} s(X, \hat{Y}).$$

Thus, the following are parameters for the model, the matrix A representing bigram compatibility, the scores needed for the matrix P, linear weights, and the word embeddings. The work not only proved to show improvements in F_1 scores, but the work also became a reliable and commonly used architecture for NER tasks (Lample et al. [2016]).

2.2.4 Named Entity Recognition Evaluation Measures

It is important to be able to measure the quality of the models in any automated system. In this section, we will discuss the evaluation measures for the performance of the system. Precision and accuracy are two commonly used measures for the evaluation of a system; we use these measures to calculate the performance of the model. *Precision* is a measure that describes the number of correct positive labels, and we can interpret the precision of the model as the consistency of the model. In binary classification we define it formally as:

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

where TP and FP represent true positives and false positives, respectably. Recall is the measure of the positive classification, and is linked to the *sensitivity* of the model. We can define recall as:

$$\frac{TP}{TP + FN}$$

. Accuracy is a measure of how close the model is to the truth; in binary classification, we can define it as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TN and FN represent true negatives and false negatives respectably. Another measure that is of importance is the $F_1 score$, which considers both accuracy and precision. This measure is the harmonic mean, which takes a value between 0 and 1. When the model correctly classifies each value into the correct class we have a F_1 score, or *F-measure*, of 1 and we define it formally as:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Precision and accuracy of the named entity recognizer are traditionally measured using hand-crafted data from linguists. Outside of traditional measures, there are domain-specific measures for named entity recognition. We will briefly discuss the techniques used in main NLP conferences. The Message Understanding Conference (MUC),are one of the most commnly used standards used in NLP for evaluating information extraction systems (Chinchor and Sundheim [1993]). MUC evaluations separate the evaluations into two realms, one focuses on the *type* and the other on the *text*. As systems tend to measure several entities types at a time, the *type* focus is on matching the entity tag appropriately, for example, in the sample sentence:

Charles University is a university located in *Prague, Czech Republic*.

The first entity in the sample sentence should match the entity tag for organization and not, say, an entity for people. The second measure is based on the boundaries of the entity that is being detected. For example, if the system recognized **in Prague, Czech Republic** instead of the italicized entity in the sample, it would be penalized in its performance (Nadeau and Sekine [2007b]). Other systems simply take into account if the match is an exact match, disregarding partial matched or boundaries.

2.3 Issues in Named Entity Recognition

Now that we have a brief introduction to the task, the methods, and the evaluation of NER, it is important to also survey the issues that one may face when building and improving a NER system. One of the biggest challenges with named entity recognition and machine learning approaches is the appropriate training data. In order for machines to learn the diversity of entities and their meaning given their context, systems need to be trained on a large amount of data. Data quality is also a concern as training data needs to match the domain of interest. A large amount of data is just the first part of the issue of data quality, the annotated data must be reliable, and typically this means human linguists must spend laborious hours tagging and evaluating this data before being put into the training stage. The domain-specific training data, while improving performance in the specific domain, sheds light on the limitations of NER systems, as there is no omniscient recognizer for the myriad of tags a given text can contain.

2.3.1 Ambiguity

One of the biggest challenges of named entity recognition is named entity disambiguation. Entity disambiguation has been categorized into a machine learning task of its own, with a multitude of systems solely dedicated to disambiguation. This task can sometimes be referred to as entity linking, or when systems are combined with named entity recognition, these systems are referred to as Named Entity Recognition and Disambiguation (NERD). The issue of disambiguation brings us back to the notions of understanding the linguistic principle of context. An example sentence for entity disambiguation:

> The *Black Mamba* scored the most 3-point shots in his career during the championship game.

The entity *Black Mamba* should be linked to the famous basketball player Kobe Bryant and not to the venomous snake found in sub-Saharan Africa. There are many techniques to alleviate this issue; previous studies have tried to model *textual* context by modeling the similarities between words and a probable entity ([Yamada et al., 2016]). Other approaches use word embeddings to capture the nuances of language modeling context, which we will discuss in detail in the next chapter. In the following sections, we will discuss some ways to alleviate the issue of ambiguity in the task.

Disambiguation with Deep Learning

Disambiguation using Deep Learning combines the components of the artificial neural networks mentioned previously. The Deep Learning approaches discuss here will describe the architecture of artificial neural networks which have learned models for disambiguation or joint entity recognition and disambiguation (NERD).

In the paper *Modeling Mention, Context and Entity with Neural Networks for Entity Disambiguation*, researchers present a new model that acknowledges semantic representations of mention, context, and entity and encodes the semantic information into a continuous vector space to be used for disambiguation (Sun et al. [2015]). The task in the paper is recast as ranking by comparing the similarities of input and candidate entities. The proposed neural network is as follows, a representation for the context sentence is encoded into a vector (minus the mention) v_c as well as the mention represented as the vector v_m. The representation of the context vector v_c is modeled with a convolution neural network, whose inputs include word embedding as well as position embedding, which is one of the main contributions of the paper. This positional contribution is backed by the intuition that closer context words may be of more value in terms of encoding information about the entity. The mention vector is represented as word vectors, and when there is a multi-word mention, the average of the word vectors is calculated. The entity associated with the mention is encoded as two vectors $w_e w$ (entity word representation) and $v_e c$ (entity class representation). The entity class and entity words are created using words or phrases from the reference knowledge base and the entity, respectively. Class entities are sourced from info-boxes in the knowledge base, which are mapped in a continuous vector space. The two vectors $v_e w$ and $v_e c$ and input to a tensor to create the vector v_e, which represents the entity. Meanwhile the vectors v_m and v_c are also input into a tensor to create the vector representation of context and mention $v_m c$. The two resulting vectors are then compared for similarity using $cos(v_m c, v_e)$. This measure is then applied to the entity disambiguation task. The selection of the entity is simply calculated by the entity with the closest entity using the previous calculation. For the training of the model a loss function is defined as:

$$loss = \sum_{(m,c) \in T} max(0, 1 - sim(e, mc) + sim(e', mc))$$

Where e is the correct entity, and e' is a random entity, this is function assumes that the correct entity is larger than the similarity score of a randomly selected score from the knowledge base by a margin of 1. The results of the approach described in this paper show improvements in accuracy when integrating the semantic vectors from the previous state of the art by up to three points.

While many of the approaches to disambiguation contain an input text and an entity, some deep learning approaches use the legacy concepts of graph elements. In the paper *Named Entity Disambiguation using Deep Learning on Graphs* the model combines an input graph and runs a consistency test of the input text and the graph (Cetoli et al. [2018]). The graph is represented with nodes as with *Glove* word vectors, using averages of the word vectors in multi-word instances. The edges of the graph are also represented by averaging the word vectors in the edge's labels. The input text of the model is represented by a sequence of word

vectors, which are fed into a bidirectional LSTM, resulting in a vector that is weighted by a mask, a set of binary numbers that note the location of the entity by 1, and 0 all other indices. This is combined with a recurrent neural network to process the knowledge base triples. Using the aforementioned node vectors for the graph input and input text vectors from the LSTM model, they apply a hidden layer that produces a binary vector that provides information about the consistency between the graph and the input text. The following architecture is the first model, with successive models which increase the complexity of the model by adding an attention mechanism or by changing the RNN representation of triples into a graph convolutional network (GCN). The attention mechanism, proposed by Bahdanau et al., originally for the task of machine translation, is a mechanism that jointly learns alignment and translation by a linear combination of encoder and decoding states. (Bahdanau et al. [2014]) In this task, the output vectors of the LSTM are first weighted by the attention coefficient, then summed and used for the creation of the graph vector representation. The graph is then calculated as:

$$y_{graph} = N_{triplets}^{-1} \sum_{i=0}^{N_{triplets}} b_i z_i$$

Where b_i represents the attention coefficient, z_i represent the output vectors of the LSTM, $b = softmax(c)$, and $c_i = ReLU(\mathbf{W}_{triplets}\mathbf{z}_i + \mathbf{W}_{text}\mathbf{y}_{text} + \mathbf{b}_{triplets})$. $\mathbf{W}$ matrices and the vector b are learned during the training phase of the algorithm. The intuition of this architecture is that the attention mechanism will recognize relevant triples and place more weight on them as a result. The next approach described in the paper is one that combines graph convolutional networks, originally proposed by Kipf and Welling. Graph convolution networks are neural networks that, when given a graph, $G = (V, E)$ (vertices and edges), the network uses the input matrix, X, where $X = N * F^0$ representative of the features. The dimensions of X, where $N = |nodes|$ and F^0 is the number of input features for each node. An $N * N$ matrix A, representative of the graph structure, can be an adjacency matrix (Kipf and Welling [2016]). Hidden layers in the GCN architecture can be defined as $H^i = f(H^{i-1}, A)$, where $H^0 = X$ and f is a propagation, for example $f(H_i, A) = \sigma(AH_iW_i)$. The resulting architecture is then the bidirectional LSTM and the GCN representing the triples, as well as the same architecture with attention. Of the many architectures described in the paper when analyzing the precision, recall, and F_1 score, the best performing model overall was the Text LSTM and RNN of a triple with attention and the same model without the attention mechanism performing slightly worse. All of the aforementioned models focus only on the task of entity disambiguation, but there are many studies that attempt to jointly solve the recognition and disambiguation of entities.

In the paper *A Novel Ensemble Method for Named Entity Recognition and Disambiguation based on Neural Network*, the method, called **Ensemble Nerd**, combines the responses of NERD extractors and employs two Deep Learning networks to carry out the task (Canale et al. [2018]). The first of the two is an ENNTR (Ensemble Neural Network for Type Recognition). This is done by concatenating several inputs from the multiple NERD extractors. These features include type features, score features, entity features, and surface features. Surface form features are relative to the input text, which is split into tokens and assigned

a word embedding computed using a pre-trained model. Type features represent the entity type such as person, place, or organization and are mapped through one-hot encoding. The features of the entity are used to identify similarities between Wikidata entities; this is done using a vector with several dimensions. The first being a boolean to represent if the entities have the same URI, the second uses Levenshtein distance to compare the labels of the entities, the third represents the TF-IDF Cosine similarity, the fourth specifically for the entity person represents if the compared entities share the same *occupation*, the last dimension represents the structural similarity in the two entities. The simplest feature, score features, are simply the confidence or saliency scores returned from each of the entity extractors. The features are generated from each of the extractors, are used as input to a dense layer, and finally into an output layer. The second network, ENND (Ensemble Neural Network for Disambiguation), differs from the latter and attempts to determine whether the result from one of the extractors in the collection is correct or not. This is done using an architecture that also concatenates features from entity similarity and type features from the extractors. The type features are similar to the previous network type features and are then input to two dense layers to a single neuron output layer, with a sigmoid activation function. The importance of this work is that its approaches differ from the Bi-LSTM approach and shift the perspective of neural network architecture.

3. Word Embeddings

Neural networks have brought an array of new ideas and improvements in the state-of-the-art performance in Natural Language Processing tasks. The models previously discussed are similar as they use representations of the input language. These representations, called word embeddings, have also been evolving alongside the algorithms that use them. This thesis will also use several varieties of word representations, and this chapter lays the framework of the pre-existing word representations as well as the ones that are used in the practical implementation of this thesis.

In its most simplistic definition, word embeddings are numerical representations of text. One hot encoding offers a straightforward representation of input text. For the input sentence *The quick brown fox jumps over the lazy dog*, each of the words in the language is mapped to a binary vector representing a word in the language. We do this mapping by collecting all of the words in the language and represent the index of the occurrence of the word as a true value and the rest of the indices of the dictionary as false values. This results in a vector $[0, 0, 0, 1, 0]$ for each of the words. This representation of words, while straightforward, has its limitations. Say that you want word embeddings to capture the essence of the word and be able to map similarities between words and concepts. This idea of identifying similar words is discussed in distributional semantics; this is driven by the semantic theory that words used in similar contexts tend to have similar meanings (Sun et al. [2015]). This is one of the many linguistic theories that has been modeled further in the domain of NLP. Representing words as vectors isn't limited to the input of neural network models. Words as vectors have been used in information retrieval tasks to model documents and queries for search engines. This approach is very similar to the one-hot encoding mentioned above.

One of the most significant algorithms for modeling word embeddings is the *Word2vec* model. The word embeddings returned from the *Word2vec* model are considerably different from the one-hot encoding. The algorithm provides a distributed representation of words. This means that instead of the vector dimensions representing the size of the dictionary N, it has a fixed vector length V where each of the elements in the vector provides information about the words. The paper *Efficient Estimation of Word Representations in Vector Space* offers new techniques for distributed word representation and does so in two ways (Mikolov et al. [2013]). The paper explores two neural network architectures: CBOW (Continuous Bag-of-Words) and Skip-grams. The models adopt an architecture similar to a language model, modeling the context of the words by their surrounding neighbors. In the CBOW architecture (in figure 3.1 below), the model loads words preceding and following a given word in a certain time window (for example, two words before and two after). Each of the context words is encoded into one-hot vectors (thus, if the size of the dictionary is N, we have N dimensional vectors). The context vectors are propagated to a single hidden layer and then to an output layer. The skip-gram model is theoretically the inverse of this CBOW architecture. Instead, it uses a single representation of a word (again one-hot vector) which is propagated to a hidden layer, and target context words are represented in the output layer. While the work done by Mikolov et al. is comprised

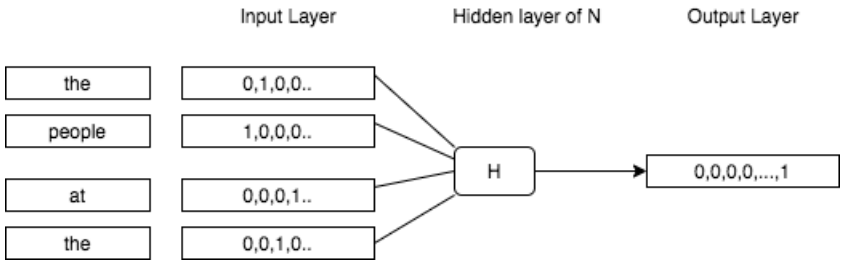

Figure 3.1: Architecture of Continuous Bag of Words network, adapted from Mikolov et al. [2013] Figure 1.

of seemingly simple architectures, the results of their work are surprising. The *Word2vec* embeddings can capture linguistic relations such as countries and their capitals, male and female representations of words (for example, brother/sister), comparative adverbial relations (such as small and smaller), verb tenses (walk and walking). This is accomplished using arithmetic operations with the vector representations. The Word2vec model offers hope for word representations, as it captures semantic and grammatical relations of words. When evaluating the overall performance of the model, it is important to note the disadvantages of the model. In some of the semantic tasks, the logical analogies are not intuitive. Rather they are indicative of the training data and their contextual bias, for example, *king:queen::man:* returns *woman* as expected, but also returns *Attempted abduction*. Word2vec has its advantages as it is open source and quite easy to access; on the other hand, it cannot handle out of vocabulary words (OOV). There are also no shared representations of subword units (such as inflectional information) encoded into the vectors. Subwords (smaller units of words) can prove very useful for handling OOV words and have also proved to be an interesting approach for word embeddings. Another difficulty with the Word2Vec's model is its inability to handle ambiguity; for each word, there is one vector representation. In the next sections, I will outline some alternatives to Word2vec and analyze in-depth the architecture of those used for this thesis work.

GloVe

The work on word embeddings led to breakthroughs that aimed to surpass the performance of the Word2vec. Word2vec remains a popular option for NLP applications, but there are a few other contenders that also perform on the same level and leverage other advantages. One of which is the *GloVe* shorthand for Global Vectors. GloVe was developed by Jeffrey Pennington, Socher, and Christopher D. Manning, members of the Computer Science Department at Stanford University. Its NLP department continues to provide industry-standard tools for NLP, such as the taggers, classifiers, and parsers. Much like Word2Vec, GloVe embeddings derive the relations between words by using cosine distance. This makes the above example about semantic relations between words such as *queen::king* also applicable to the GloVe embeddings. GloVe embeddings also have the feature of having words that are semantically similar being near to each other in the vector space; thus, looking for k nearest neighbors of any word can result in synonymous or similar themed terms. GloVe embeddings also feature the ability to use

arithmetic on the embeddings to get a sense of the meaning of the words. Thus we can derive the *Country:Capitol*, *singular:plural*, and gendered representations of words using some vector expressions.

The main difference between the Word2vec embeddings and the GloVe embeddings is their training methods. In the Paper *GloVe: Global Vectors for Word Representation* by Pennington et al., they aim to identify the *semantic and syntactic regularities* which didn't have an apparent explanation of how they are produced in the previous Word2vec model (Pennington et al. [2014]). The method used for GloVe tries to take into account the global co-occurrences and not just the local context as the former model does. To understand the logic used, we must first examine how the methods of global matrix factorization differ from the skip-gram and CBOW models. The global matrix factorization model was first developed by Deerwester et al., originally for indexing and information retrieval. In their paper *Indexing by Latent Semantic Analysis*, they identify the need for users who are querying documents to have document retrieval based on the conceptual topic being queried rather than the exact words in the query (Deerwester et al. [1990]). The pre-existing models used a matrix that has documents along the y axis and the lexicon of terms along the x-axis. This is a simplistic way to represent the documents, as one can just mark the terms in the document using a binary value of 1 in the column where the term is present and 0 a zero when not present. Simplistically this representation would return the relevant documents only if they contain the exact wording of the query document. Deerwester et al. aim to gain an understanding of *latent semantics* i.e., what the query means. Deerwester et. al use a document term matrix similar to the one above, but whose rows correspond with the terms and the documents to columns, with count occurrences instead of binary values. This structure coupled with the Singular Value Decomposition (SVD) model aims to explore the underlying latent structure. The SVD model *decomposes* the *term* x *document* matrix into a product of three matrices

$$X = T_0 S_0 D_0'$$

, where both T_0 and D_0 are orthonormal (perpendicular along a line) and the matrix S_0 is diagonal. (Deerwester et al. [1990]). The aim of the SVD is to reduce and generalize the large matrix described above. The components of the SVD model have the following dimensions:

- $T_0 = t$ x m
- $S_0 = m$ x m
- $D_0' = m$ x d,

where t represents the number of rows in X, d is the number columns, and m is the rank of X ($\leq min(t, d)$) (Deerwester et al. [1990]). The matrices T_0 and D_0' are unitary thus $\mathbf{T}_0^\top T_0 = T_0\ \mathbf{T}_0^\top = \mathbb{I}$. S_0 is non-negative and in decreasing magnitude, i.e. $S_0 = [\sigma_1 \geq \sigma_2 \geq \sigma_3 ... \geq 0]$. We can refer to T_0 as the right singular vectors, and D_0 as left singular vectors. S_0 is also also in the order of importance thus the first column of S_0 has more importance that next and so forth, we can reduce the matrix and take only the top k and set the rest to 0.

This reduction of the matrix is then applied to vectors T_0 and D_0 resulting in a new X. The updated matrix is defined as:

$$X' = TSD$$

, where $X' = t$ x d, $T = t$ x k, and $D = k$ x d (Deerwester et al. [1990]).

The SVD model lays the groundwork on transforming a large sparse vector into a more compressed vector while maintaining significant information. In matrix factorization methods for word embeddings the matrix representation is not of document and term but of terms and terms, representing the co-occurrence of terms. Pennington et. al. define the matrix X, where the entry X_{ij} represents the number of time the word j occurs in the context of word i. They define $X_i = \sum_k X_{ik}$ representing the amount of times any word occurs in the context of i, and $P_{ij} = P(j|i)$, representing the probability that the work j occurs with the context of the word i, and $P(j|i) = \frac{X_{ij}}{X_i}$ (Pennington et al. [2014]). Instead of using the raw probabilities for the co-occurrence matrix the authors introduce *probe words* k. The intuition behind modeling the matrix this way is that these ratios can better distinguish relevant terms from irrelevant ones, they generalize the model as follows:

$$F(w_i, w_j, w'_k) = \frac{P_{ik}}{P_{jk}}$$

, where w represents word vectors and w' represents context word vectors (Pennington et al. [2014]). The model is refined to take advantage of linear operations and to enable an ease of intake into a neural network the authors take the dot product of the components:

$$F((w_i - w_j)^\top w'_k) = \frac{P_{ik}}{P_{jk}}$$

. The authors note that the choice word vectors and context vectors w and w' are arbitrary and the model should be *symmetric* for $w \longleftrightarrow w'$ and X^T, in order to make this possible they require F to be a *homomorphism* between $(R, +)$ and $(R+, \times)$:

$$F((w_i - w_j)^\top w'_k) = \frac{F(u_i^T w'_k)}{F(u_j^T w'_k)}$$

. The authors imply that the F is the exponential function, thus

$$w_i^T w'_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

, the last logarithm is independent from k and can be changed to a bias b_i, in order to maintain symmetry of the model a second bias b_k is added, updating the model to:

$$w_i^T w'_k + b_i + b'_k = \log(X_{ik})$$

. The model has made a number of changes to account for the data but still has the issues of divergence of the logarithm whenever the argument is 0, as well as weighting all of the co-occurrences equally, this would be a major drawback as rare word combinations create noise in the model and don't enrich the model with much information. Pennington et. al propose a weighted least squares model that

aims to correct these problems, they introduce a weighting function $f(X_{ij})$ and define the model as:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T w_k' + b_i + b_j' - \log X_{ij})^2$$

, where V = size of the vocabulary and f is a weighting function which needs to be 0 when $X_{ij} = 0$ to ensure that the logarithm is defined well:

$$f(x) = \begin{cases} (\frac{x}{x_{max}})^\alpha & if x < x_{max} \\ 1 & else \end{cases}$$

, they define the cut-off parameter $x_{max} = 100$ and $\alpha = 3/4$, thus the logarithm of the co-occurrence matrix X is implicitly factorized. The resulting word embeddings have improved performance in NER tasks but still have the issue of managing out of vocabulary words poorly. GloVe embeddings only have representations of words that have been seen in the model already; thus, when testing on new data, new words are often subscribed to a random or a null vector. In the next section, I will discuss ways to manage this issue.

3.1 Sub-word Embeddings

Sub-words are linguistic units that are building blocks for languages. In morphologically rich or polysynthetic languages, many morphemes, which can be synonymous with sub-words, are the elements with meaning that are bound to one word. In some instances, polysynthetic languages can be translated into many words in their less synthetic counterpart language, such as English. This can be problematic in word embeddings with a framework like Word2vec. The issues with representing words without any consideration of their smaller units may also present issues in languages like English still, as taking whole words irrespective to smaller inflectional units or morphological derivations: *girl* and *girls*, *hope* and *hopeful* for example have no similar underlying unit in this model). The work of Sennrich et al. in *Neural Machine Translation of Rare Words with Subword Units* demonstrates an algorithm for word segmentation, called *byte pair encoding* (Sennrich et al. [2016]). Byte pair encoding, originally proposed by Philip Gage for data compression (Gage [1994]), tokenizes subword units in the following manner:

1. Initialize the dictionary with all of the characters.
2. Represent each word as a sequence of characters with a special symbol to represent the end of the word.
3. Iteratively count pairs of characters in all of the words.
4. Merge the most common pairs and add them as new entries to the dictionary.
5. Repeat the merge operation until several merges have been met or a predefined dictionary size is reached.

The experiments conducted by Sennrich et al. confirmed that subword units generated intelligently help improve the performance of machine translation and create a framework that is closer to open vocabulary by representing words as a sequence of subword units.

Character Embeddings

Word embeddings and subword units aimed to find methods to handle language in small units; character embeddings ventures further into the smaller units of language and simply use characters. In the paper *Text Understanding from Scratch*, Xiang and Yann introduce a neural network that uses characters instead of words (Zhang and LeCun [2015]). The study applies temporal ConvNets to several NLP tasks, using characters as inputs and abstract properties of the word meaning as output. Xiang and Yann hypothesize that the models are "learning from scratch" as the input lacks the knowledge of larger linguistic components (words), as well as syntactical and semantic structures. ConvNets are convolution networks that are typically used in computer vision. The architecture (with images as input) uses three types of layers to build the network. Similar to previously discussed architectures, ConvNets typically use an input layer (with images in mind this will represent the raw pixel values), a convolutional layer to extract local features of the image, a RELU layer (a max function(x,0)), a pooling layer for down-sampling, and a fully connected layer. Xiang and Yann introduce a new convolutional module specifically for the understanding of text. The component computes a 1-D convolution between the input and output, first they suppose a discrete input function $g(x) \in [1, l] \to \mathbb{R}$, and a discrete kernel function $f(x) \in [1, k] \to \mathbb{R}$, and the convolution as $h(y) \in [1, \lfloor (l-k)/d \rfloor + 1] \to \mathbb{R}$ between $f(x)$ and $g(x)$ with a stride d defined as:

$$h(y) = \sum_{x=1}^{k} f(x) \cdot g(y \cdot d - x + c),$$

they define $c = k - d + 1$ is used as an offset constant, (Zhang and LeCun [2015]). Model has a set of kernel functions to use as parameters of the model $f_{ij}(x)$, where $i = (1, 2, 3, .., m)$ and $j = 1, 2, 3, .., n$, which are referred to as weights on the set of inputs. The outputs $h_j(y)$ are calculated by summing over i number of convolutions between the input and $f_{ij}(x)$. The model also differs from the traditional convolution networks as it uses temporal max-pooling; this allows them to have a 1-D pooling module. The model architecture accepts a sequence of words as encoded characters as input which is then encoded into vectors. This is done with a representation of the alphabet of size m and representing each of the characters as a 1-of-m vector. The vectors are then concatenated and transformed to a larger vector of a fixed length l; anything that doesn't fit into the fixed size is simply clipped. The model is then tested on various tasks, including sentiment analysis, classification, with results that support that character embeddings are a promising approach to representing words.

The idea that words along with their word components carry meaning sparked different approaches in architectures of character embeddings. The CNN approach previously discussed introduced the benefits of the character embeddings, but intuitively language problems are not spatial tasks; rather, they are sequential tasks. This is due to the nature of language as a sequence. While this

doesn't make the CNN approach wrong, rather, it provides a flip in perspective and some promising results. The CNN architecture has resulted in some improvements in NLP classification tasks, but the convolution and pooling information may allow the model to forget information about the order of the input, which is quite critical to NER and other NLP tasks. In the paper *Comparing CNN and LSTM character-level embeddings in BiLSTM-CRF models for chemical and disease named entity recognition*, by Zhai, Nguyen, and Verspoor, a thorough comparison is conducted on the two for NER in the biomedical domain (Zhai et al. [2018]). The architecture for the LSTM based character-level word representations follows that from (Lample et al. [2016]). The character embeddings correspond to the characters in the word and are input into a forward and backward LSTM to represent the words from left to the right. This representation from both left and right is then concatenated with a word representation from a lookup table. Pretrained word embeddings are used as the initial scores lookup table, which uses the skip-gram method, which is then fine-tuned during training. If the word is not found in the lookup table, then a UNK embedding is used. The intuition of using both forward and backward representations of the word is to evoke the bias of the LSTM's most recent inputs. Thus the forward LSTM provides a better representation of the suffix of the word, and the backward LSTM better represents the prefix. Dropout training is used to improve the model as well as to create a dependency on both character and word representation. The dropout mask is applied to the final layer before going into the BiLSTM NER model (the architecture is discussed in Chapter 2).

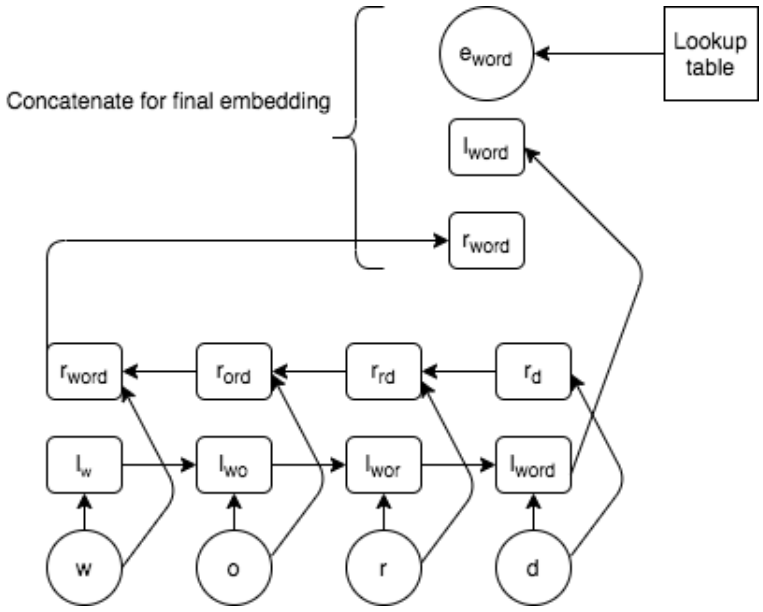

Figure 3.2: Architecture of BiLSTM with character embeddings, adapted from Lample et al. [2016]

The results of this study show that both CNN and LSTM based character embeddings perform similarly, whit the performance of LSTM models slightly better. However, the study also shows that the CNN-based character embeddings have reduced complexity as the number of parameters differs immensely. The LSTM model increased the execution time of the Bi-LSTM CRF model by 115%

while the CNN model only by 25%, thus for many, the CNN model would be preferable for NER on biomedical entities.

3.2 Contextualized Embeddings

In the architectures previously discussed, contextual information was a part of the architecture of the overall model. In this section, we explore the benefits, architecture, and methodology of contextualized embeddings. Word2vec was one of the breakthrough NLP technologies; it continues to perform well on many NLP tasks. Using Word2vec, word embeddings are limited in their ability to handle the nuances of languages, mainly ambiguity. For each word in the language, the vector that corresponds to the word remains the same regardless of context. Thus the vector for *bank* in the sentence *She relied on the teller at the bank for her financial statement* and *They had to bank up the snow to create a suitable pathway* remains the same even though the work *bank* is polysemous, thus has different meanings, which can be understood based on its context. In contrast, native speakers of a given language pick up the clues of context to distinguish the meanings word embeddings typically do not. A new wave of word embeddings has since been introduced, which model context using deep learning. Some of the most notable contextualized word embeddings are BERT, ELMo, and GPT-2.

One of the recurring themes in contextualized word embeddings is the language models and how they use them for training the word embeddings. A language model represents the probability distribution over a sequence of words in a given language. Given the sequence $S = (w_1, w_2, w_3, .., w_m)$ the language model then assigns a probability of the sequence using unigrams, bigrams, or in the case of neural networks, multi-layer LSTMs. The language model task, though a simple approach, has allowed for word embeddings to *learn* the linguistic properties of words and outperforms its predecessors in many tasks.

ELMo

In the paper *Deep Contextualized word representations* by Peters et. al, ELMo (Embeddings from Language Models) are introduced as a solution to model the syntax and semantic characteristics of word use as well as its nuanced meaning differences to model polysemy (Peters et al. [2018]). ELMo differs from Word2vec in such a way that each word representation is function that uses the entire input sentence. The vectors are produced from Bi-LSTM that is trained with a language model on a large corpus. The Bi-LSTM language model (BiLM) as the authors refer to it, uses the input of a sequence of tokens $t_1, t_2, t_3, ..t_N$, where N is the number of total tokens. A forward language model is then used to compute the probability of a token t_k given its history (left tokens) of $t_1, ..., t_{k-1}$, which is computed by:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1}).$$

A context dependant token representation x_k^{LM} is pushed through L layers of the forward LSTM to create $\underset{h}{\rightarrow}_{k,j}^{LM}$, where $j = 1, ..., L$, the layer of the output L predicts the next token using a softmax layer. Intuitively the backward LM is

very similar to the forward LM, except that the sequence is process in reverse:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ..., t_N)$$

each layer of the backward LM produces a complementary representation $\underset{h}{\leftarrow}^{LM}_{k,j}$. The two language models are then combined with a formula that maximizes the log likelihood of both the language models using the following formula:

$$\sum_{k=1}^{N} (\log p(t_k | t_1, t_2.., t_{k-1}; \Theta_x, \underset{\Theta}{\rightarrow}_{LSTM}, \Theta_s) + \log p(t_k | t_{k+1}, .., t_N; \Theta_x, \underset{\Theta}{\leftarrow}_{LSTM}, \Theta_s)),$$

where Θ_x is the token representation and Θ_s is the softmax layer. ELMo specifically is a *task specific combination* of the layers in between the biLM. For each of the tokens, a biLM with L layers calculates $2L + 1$ representations:

$$R_k = \{x^{LM}_k, \underset{h}{\rightarrow}^{LM}_{k,j}, \underset{h}{\leftarrow}^{LM}_{k,j} \ | j = 1, ..., L\}$$

$$= \{h^{LM}_{k,j} | j = 0, ..., L\},$$

where $h^{LM}_{k,0}$ represents the token layer and $h^{LM}_{k,j} = [\underset{h}{\rightarrow}^{LM}_{k,j}; \underset{h}{\leftarrow}^{LM}_{k,j}]$, for each layer of the biLSTM. The model then collapses all of the layers into a single vector representation, thus $ELMo_k = E(R_k; \Theta_e)$. The selection of the layer can be done in two ways: simply by choosing the top layer or by weighing the all of layers on a specific task:

$$ELMo^{task}_k = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s^{task}_j h^{LM}_{k,j},$$

where s^{task} represent soft-max normalized weights, and γ^{task} is a scalar parameter that allows for the task model to scale the ELMo vector Peters et al. [2018].

ELMo and other deep contextualized models have outperformed the state-of-the-art static word embeddings, but understanding how these dynamic word embeddings can model polysemy and perhaps ambiguity is another question entirely. Exploring how contextual these models are can expand our capabilities and general knowledge on how to create even better representations of words by learning its drawbacks. *How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings* examines the representations of contextualized word vectors within the vector space. Surprisingly the vectors in contextualized word embeddings, and in particular ELMo context specificity increases in the upper layers and words in the same sentence begin to have more similar representations (Ethayarajh [2019]). It is important to note that in the case of ELMo, the models don't explicitly *learn* the word and all of its unique word senses. ELMo rather creates multiple representations of the token, which are quite specific; thus, the token *cat* can have multiple representations, even though this isn't entirely what speakers may find intuitive. With all things considered the author also notes that some of the words that have the most context-specific representations are: *and*, *of*, *'s*, *the*, and *to* (Ethayarajh [2019]). ELMo and other word embeddings continue to improve NLP downstream tasks, including those of domain-specific named entity recognition.

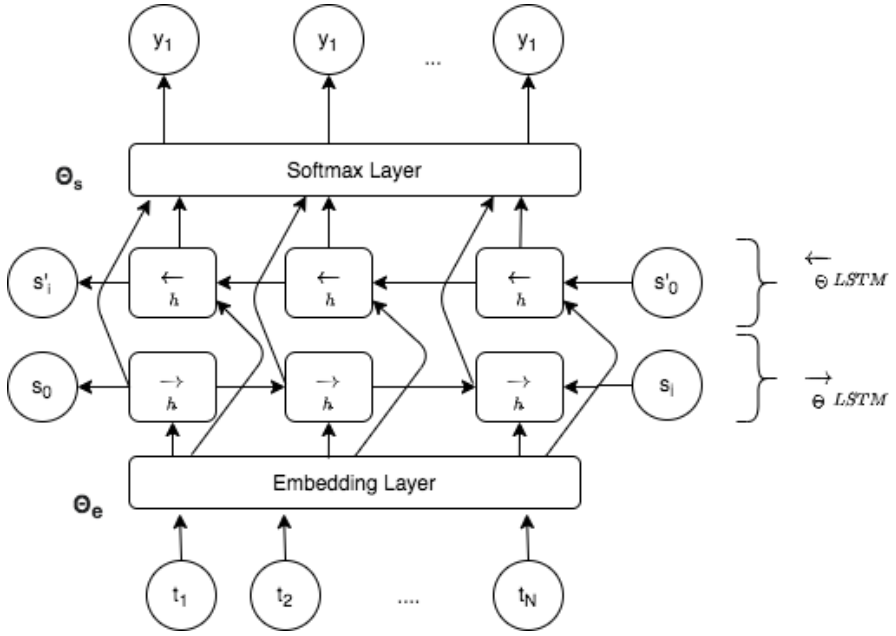

Figure 3.3: Architecture of ELMo

BERT

ELMo's representation of language rivals that of BERT (Bidirectional Encoder Representations from Transformers), developed by researchers at Google AI Language. The aim of the researchers was to provide improvements in the fine-tuning of pre-trained language models which use a feature-based approach (ELMo) or a fine-tuning approach (GPT). BERT aims to exploit the bidirectional properties of the text and differs from ELMo in this manner. Recall the last section, which explains that in order to consider both left and right contexts of the words, ELMo uses right-to-left LSTM as well as a left-to-right LSTM and concatenates them to represent left and right context. BERT is able to do so using two defining feature transformers and a masked language model.

BERT's use of a transformer model as opposed to the LSTM model has made it stand out to the later models as it makes use of parallelization of the tasks associated with sequence to sequence models. In the latter LSTM model, each step in the sequence accounts for a step in time, making this process very time-consuming for larger data sets. The original transformer model proposed by the Google Research Team in their paper *Attention Is All You Need*, describes the need for such an architecture (Vaswani et al. [2017]). The transformer architecture has two key features the encoder and the decoder. The encoder encodes the inputs and digests an input embedding as its first step. This embedding is the representation of the entire sentence, not just a singular word. In order to retain some positional information about the items in the sequence, Vaswani et al. use a positional encoding mechanism in both the inputs and outputs. The position encoding has the same dimension of the embeddings in order for them to be summed, and the following two functions are used:

$$PE_{pos,2i} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{pos,2i+1} = \cos(pos/10000^{2i/d_{model}})$$

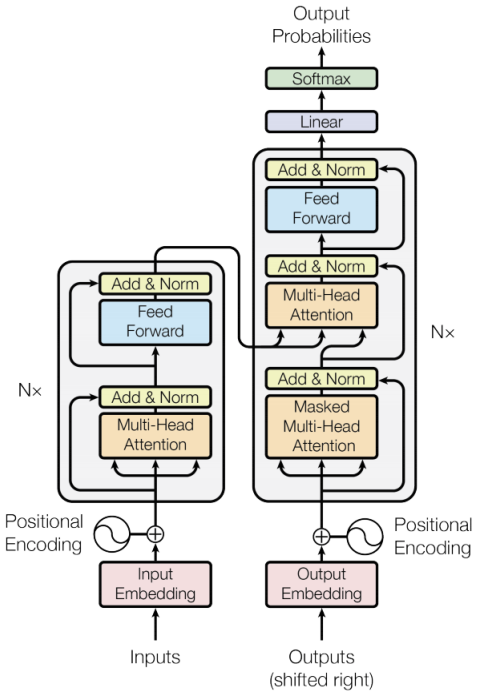

Figure 3.4: Transformer Architecture (Vaswani et al. [2017], Figure 1)

, where *pos* represents the position and i represents the dimension (Vaswani et al. [2017]). The encoder stack has two additional components the multi-head attention mechanism and a position-wise fully connected feed-forward network. The multi-head attention function is a critical part of this architecture and is built on the basis of an attention function. Vaswani et al. describe the attention function as a mapping from a "query and a set of key-value pairs" to an output, nothing that all of the values are represented as vectors. The attention aims to provide the context needed for each of the subunits of the sequence since the inputs are ingested at the same time, unlike LSTMs or RNNs. The attention blocks generate attention vectors for each of the units in the sequence, representing the context or how relative one word is to another in the sequence. The singular attention mechanism in Figure 3.5 (Vaswani et al. [2017]) The inputs Q is a matrix representing a set of queries, V and Q are keys and value matrices respectively and are computed as

$$Attention(Q, V, K) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

, (Vaswani et al. [2017], pg. 4). The multi-head attention extends the logic of the single head attention but aims to "linearly project" each of the inputs h times with different linear projections d_k, d_q, and d_v dimensions (Vaswani et al. [2017], pg. 4). The attention function is performed in parallel to the newly projected versions of the matrices, attending to different positions in the sub-space at once. The other distinguishing feature of the transformer model is the masked multi-head attention. The masked multi-head attention is similar to the multi-head attention above, but instead of getting all of the items in the input sequence, it

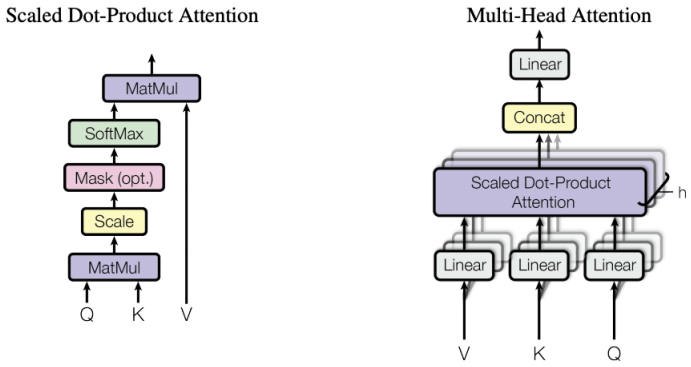

Figure 3.5: Scaled Dot Product Attention and Multi-Head Attention (Vaswani et al. [2017], Figure 2)

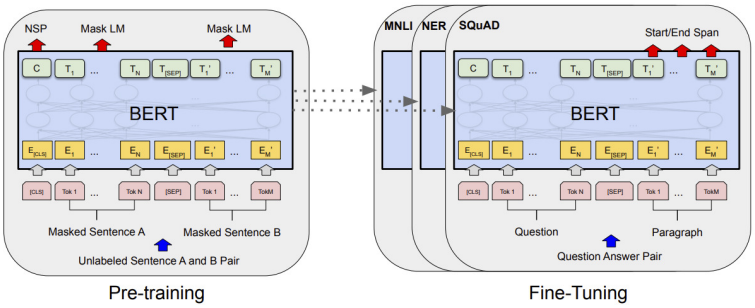

Figure 3.6: BERT Architecture (Devlin et al. [2018], Figure 1)

receives only the previous units of the sequence, and the units after are *masked* and represented as zeros. This mechanism prevents the model from computing the attention vectors for the upcoming words at each step.

This masked approach is important in the BERT model as it makes use of a *masked* language model based on this logic. This is done by randomly masking a percentage of the input tokens in an effort to predict the original token only using its context. BERT also is tasked with "next sentence prediction", which jointly pre-trains text-pair representations (Devlin et al. [2018], pg. 2). BERT has two main steps, pre-training and fine-tuning. During the pre-training, the model is trained using unlabeled data over a series of different tasks. Fine-tuning entails initializing the model with pre-trained parameters and then fine-tuning the parameters based on the particular downstream task. The architecture is similar to the transformer architecture described in Vaswani et al.'s *Attention Is All You Need*, and distinguish the base model as $BERT_{BASE}$ and the larger model as $BERT_{LARGE}$. The base model is defined as $(L = 12, H = 768, A = 12, P = 110M)$, and the large model as $(L = 24, H = 1024, A = 16, P = 340M)$ where L represents the layers (transformer blocks), H is the hidden size, A the number of attention heads, and P the total number of parameters. The input of the model can be both a sentence or a pair of sentences (for the question answering task). WordPiece embeddings

are used as the input representation of the text. They are much like BPE and segment words into subwords, i.e., *playing* $\rightarrow$ *play* and *##ing*. Two new special tokens are added to the model ([CLS]) denoting the start of the input sequence, ([MASK]) for the masked tokens, and ([SEP]) for separating the sentences in the sequence. The final input representation is expressed as a sum of the token, segment, and position embeddings. This representation of the data coupled with the transformer architecture and attention mechanisms allows for a model that has a performance that exceeded state-of-the-art performance in several tasks. One of the tasks where a significant improvement in performance was the General Language Understanding Evaluation (GLUE), where $BERT_{LARGE}$ outperforms ELMo by 11% (82.1% compared to ELMo's 71.0%) (Devlin et al. [2018], pg. 6). The ability and diversity of word embeddings are continually changing in the advancement in the NLP field; this allows us to continue to explore and improve the performance of many NLP tasks. In this section, I focus on the most critical at the moment and lay the groundwork for the representations used in the experiments described in the next chapter.

4. Methodology

This chapter will describe the data sets, experiments, and methods used for the practical work of this thesis.

Motivation

During the start of this thesis I found that in my professional work in the biomedical domain I had to access vendors which were offering IE tools. Often times these tools did not use machine learning approaches, rather they focused on specially curated ontologies and used these as the base for the dictionary based approach to named entity recognition. Through this work I grew knowledgeable about the industry standard of ontologies and the vast amount of resources available in the biomedical domain as reference data. Initially I wanted to use only biomedical "gold" standard data for training the neural networks, but found that while there were many sources, even the most reputable contained about 400,000 words (Kim et al. [2003]). I wanted to explore if I could use a carefully curated dictionary from ontologies that were industry standard to create training data from large data sets that are not annotated but publicly available to be used for a neural network as training data.

4.1 Modules and Tools

To run the experiments related to this thesis several tools and packages were needed. These tools were critical for running the experiments and allowed me to use pre-existing components to build the architectures described in the above chapters. This subsection will describe the tools and modules used for the experiments described in the later sections.

Python

All of the code associated with this thesis is written in Python. Python is a programming language that is easy to work with and relatively fast. Its performance in speed is not that of C++ or Java but due to widespread use and its number of libraries I chose this programming language. Python is well documented and is used frequently for NLP tasks.

Keras

Keras is a machine learning and deep learning framework and a library in Python that allows for the implementation of machine learning algorithms. It is one of the most common deep learning frameworks and is running on top of another machine learning framework TensorFlow. Keras allows for execution on GPU, CPU, and TPU, which is one of the reasons I chose to use this library for the Bi-LSTM model as well as the Bi-LSTM Character model. Keras is also well documented and provides examples for some of the most popular deep learning architectures more information on such tutorials can be found at https://keras.io/guides/. Keras has built-in modules for LSTM, Bidirectional layers, CRF layers, as well as all

of the other components necessary to build a Bidirectional LSTM with a CRF layer. The parameters used for the models are described in depth below.

XML

To read the sources in from the XML format another built-in Python library *xml*. Specifically, I used the ElementTree API to allow for a smooth process of parsing the source files used for training data.

SpaCy

To have the data formatted for the models below the data needs to be pre-processed and enriched with additional data. To run tokenization, sentence splitting, and POS tagging I used the Python library SpaCy. Spacy is an industry-standard NLP library, it allows for out-of-the-box models for named entity recognition, text classification, and linguistic processes for NLP pipelines. Namely, SpaCy is linguistically motivated and relies on rule-based models for its linguistic processes, it is comparable to standard libraries such as *nltk*.

Pronto

Pronto is another Python library licensed by MIT for browsing, creating, and parsing ontologies in OBO and OWL formats. I chose to use this parser for the knowledge bases in this thesis because it implements specifications for the Open Biomedical Ontologies, which are the relevant ontologies for this thesis. More information about the format and the Open Biomedical ontologies can be found at http://owlcollab.github.io/oboformat/doc/obo-syntax.html.

Google Colab

One of the most critical components of this thesis is Google Colab. All of the experiments in this thesis were run in Google Colab. Google Colaboratory (Colab) is a Google product from the Google Research team. It allows for the execution of Python code as well as a connection to the Google-provided storage drive. Colab functions very much like an interactive Python notebook but stands out in its ability to perform due to its access to GPU and TPU hardware. All of my experiments were run using the GPU virtual machines as training neural networks would otherwise be very time and space-consuming.

Flair

Flair is an NLP framework developed by the Zalando Research group for state-of-the-art language models, sequence labeling, and text classification. Flair has developed their embeddings as well which can be used and *stacked* with BERT, ELMo, and other word representations. In the background, this Python library has access to another critical NLP library called *HuggingFace* which is a large repository of pre-trained models which organization such as Google AI, Allen Institute for AI, and other organizations that are top contributors to the NLP community. This framework aims to allow for word embeddings to be combined with minimal effort (Akbik et al. [2019]). Frameworks like Keras and TensorFlow

tend to focus on one of the embeddings for a given model architecture, and combining them requires many changes to any specific model. Flair was particularly attractive for this thesis for this reason. The experiment section below on ELMo and BERT describes the model architectures in-depth used by this framework.

4.2 Sources

4.2.1 PubMed

The Data used for this thesis was sourced from the PubMed API. PubMed is a database that contains over 30 million abstracts of biomedical and scientific nature. PubMed is comprised of three components, MEDLINE, PubMed Central, and Bookshelf. MEDLINE is the U.S. National Library of Medicine database that contains journal articles of life sciences, MEDLINE is unique as each of the records has Medical Subject Headings (MeSH), which can be used when querying the PubMed API. PubMed Central, an archive of texts of the biomedical domain at the National Institutes of Health's National Library of Medicine. The last component of PubMed is the bookshelf, which is a collection of books and documents in the healthcare domain, this isn't used for this thesis as it focuses on abstracts. PubMed is accessible via an API but is also available for download bulk data sets from their website.

Data Preprocessing

Due to the need for large training sets, bulk downloads were used for this thesis. The downloads are in the format of compressed eXtensible Markup Language (XML). The XML representation allows for smooth traversing of the elements of the XML file, it includes information about the authors, the journal, the country it was written in, metadata regarding the central topics or themes of the paper, the abstract, as well as the previously discussed MeSH data. A sample of the input XML is below.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE PubmedArticleSet SYSTEM "http://sample_pubmed.link">
<PubmedArticleSet>
  <PubmedArticle>
    <MedlineCitation Status="MEDLINE" Owner="NLM">
      <PMID Version="1">1</PMID>
      <DateCompleted>
        <Year>2002</Year>
        <Month>01</Month>
        <Day>16</Day>
      </DateCompleted>
```

The code sample above is simplified, but the XML may allow for nested structures, lists, and missing elements. To prepare the data for training, many of the elements in the original tree are ignored, and we focus on two elements the element that represents the abstract and the element that represents the title of the article. Full articles are difficult to access and limit the number of documents that we

can access, thus the abstracts are sufficient for the experiments. To traverse the XML library described above, and ran the pre-processing of the raw text data to split the sentences, tokenize, and get the appropriate part of speech for each word.

The resulting processes produced two data sets for training, the disease, and the gene tab-separated table. The data for diseases consisted of 59,714 sentences. This included 1,151,798 tokens, of which there were 8,240 tokens identified as diseases, and 1,143,558 tokens identified as non-diseases. The gene training data consisted of the same number of sentences but 1,135,267 tokens, of which 1,119,461 were identified as non-entities, 6,835 were identified as biological processes, 5,063 were recognized as molecular functions, and 3,908 were recognized as cellular components.

4.2.2 NCBI Disease Corpus

The NCBI Disease corpus is a corpus that is a collection of PubMed articles that are annotated with disease name and concepts (Dogan et al. [2014]). It contains 793 PubMed abstracts that were manually annotated. This data set is used for the evaluation of the disease models that were a part of the experiments below. The data set was available as a part of the data repository for the paper *A Neural Network Multi-task learning approach to Biomedical named entity recognition* and is available at the link here https://github.com/cambridgeltl/MTL-Bioinformatics-2016/tree/master/data/NCBI-disease-IOB (Crichton et al. [2017]). This corpus allows for an appropriate assessment of the model performance.

4.3 Ontologies

To reduce the manual work of tagging entities of interest, the process of tagging the entities was automated using the ahocorasick library in Python. To process the ontologies from OBO format to a simplified representation the pronto package described above is used. I also used another Python library that simply implements the Aho-Corasick algorithm, a string matching algorithm that locates instances of the entities of interest using a trie-like structure, which has increased speed over the look-up table. To build the Gazetteer for tagging the entities ontologies are used as the knowledge base. An ontology is a set of concepts within a particular domain, they consist of instances of objects, attributes, classes, and allows us to manage knowledge in a structured way. For the work of this thesis, we focus on two classes of entities, genes and their classes (mainly biological processes that interact with genes, and anatomical locations), and diseases. Gene detection and genomics have been a field that is advancing rapidly, and our understanding of genes and their interactions continues to be an area of interest in the biomedical domain. Disease detection is typically done by ontologies or knowledge bases due to their nature of being a relatively closed class of entities, but can often be incorrectly identified, for example, *AIDS* and *aids* whose casing changes the meaning entirely. Two ontologies are used for this thesis, the Gene Ontology and the Human disease ontology (DOID) (Schriml et al. [2018], Ashburner et al. [2000], Consortium [2020]).

Gene Ontology

The Gene ontology contains three types of *classes*, (concepts which interact with other concepts in the ontology):

1. Molecular functions - this class describes activities that operate at a molecular level and are performed by gene products some examples of molecular functions are *transporter activity* or *protein kinase activity*

2. Cellular components - this class is relative to cellular structures and locations of the cell where a gene performs a function, an example would be *mitochondrion* or

3. Biological Process - this class describes the processes carried out by molecular activities, an example is *DNA Repair*

The Gene Ontology not only offers the terms from the classes mentioned above, but metadata about the terms, including the relations between different entities of the ontology. This allows for us to have a graph like representation of the terms, with a loose hierarchy of *parent* and *child* terms.

```
[Term]
id: GO:0000048
name: peptidyltransferase activity
namespace: molecular_function
def: "Catalysis of the reaction: peptidyl-tRNA(1) +  .."
synonym: "peptidyl-tRNA:aminoacyl-tRNA N-peptidyltransferase .."
xref: EC:2.3.2.12
xref: MetaCyc:PEPTIDYLTRANSFERASE-RXN
xref: Reactome: ..
is_a: GO:0016755 ! transferase activity, transferring amino-acyl
relationship: part_of GO:0006412 ! translation
```

In the above sample (slightly altered with ellipsis to fit on the page), a term with all of the metadata as it is represented in the ontology. The term contains a specific ID from the ontology, is enriched with synonyms, references to external ontologies, and a descriptor providing more information about the term. This ontology is easily accessible but also under a constant state of revision by the scientific community and is regularly updated by domain experts.

Human Disease Ontology

The Human Disease Ontology (also known as DOID) is an ontology that focuses on standardized terms for human disease. The DOID integrates MeSH terms, the National Center for Biotechnology(NCBI) thesaurus, SNOMED (Systemized Nomenclature of Medicine), and OMIM (Online Mendelian Inheritance in Mane). Much like the Gene Ontology, the DOID defines relations, attributes, and disease concepts by using a knowledge network to represent the semantic relations between the entries. A sample entry in the ontology is seen below.

```
[Term]
id: DOID:0050672
```

SENT_ID	WORD	POS	TAG
185	After	ADP	O
185	flash	NOUN	O
185	photolysis	NOUN	BIO_PROCESS
185	of	ADP	O
185	the	DET	O
185	CO	PROPN	O
185	derivative	NOUN	O
185	,	PUNCT	O

Figure 4.1: Sample Input of formatted sentences

```
name: dyskinetic cerebral palsy
def: "A cerebral palsy that is caused by damage to the  .."
synonym: "Athetoid Dyskinetic Cerebral Palsy" EXACT []
is_a: DOID:1969 ! cerebral palsy
```

Both of these ontologies share the same OBO format, with entries being defined as terms and generally have to the same structure. Due to its rich structure, only the term name and term synonyms are used as input for the gazetteer. The input data is representative of the keywords is also modified so that they exclude noise. This is done using a list of common words in English generated by Google, more information about this list can be found at https://github.com. Some of the entries that could lead to noise in the training model are removed. I also kept track of the words removed to make sure that no entities of interest were being deleted from the model. It is important to note that to track overlapping entries, the sting matching method only considers the longest matching entry as a match for example if the sentence contains *lung cancer* the string matching algorithm will only tag *lung cancer* and not tag both *cancer* and *lung cancer*. After being processed and tagged the entries started as compressed XML the input data are converted into CSV format.

4.4 Experiments

Several different models were implemented using PubMed abstracts as the input data, the first architecture if the bidirectional LSTM model with a CRF layer and simple word embeddings, the second is an LSTM architecture with character embeddings generated from an LSTM, and the last experiments use a BiLSTM network with ELMo and BERT word embeddings, (separately not combined). Each experiment is run twice to represent each of the ontologies that were discussed earlier. The models consist of implementations from the first two experiments in Keras and the latter in Flair. The intuition is that while BiLSTM may show promising results the experiments with character embeddings and contextualized word embeddings will improve the model and its ability to handle OOV words and ambiguous terms. The input data for the Keras model was first split into two data sets with the test representing 20% of the data, the resulting data is then split so that the validation set represents 25% of the test set, resulting in

train, test, and validate (this is done simply by using the sklearn library). For the Flair models, they require their training, testing, and validation to be split before running the model, thus I split the data in half using the first half for training and the second half was further split for testing and validation, respectively. The training, testing and validation set for the Flair models are 25,034, 27,434, 4,328 for the gene model (respectively) and 38,860, 29,016, and 21,532 for the disease model. Initially I was tracking the overall performance using accuracy, but this measure can be misleading, for example one model performed an overall accuracy of 98%, but on the classes it performed in the 60th percentile, because most of the words are not entities it was weighting this score highly. Below I report the accuracy during training as the other measures aren't explicitly available during training, but for the overall performance precision, recall, and F1 are used as they are more descriptive on the model performance on each of the entities. All of the code can be found on github at this link https://github.com/shadasha-will/thesis, note that none of the source PubMed articles are here, they are rather large and can be found at https://ftp.ncbi.nlm.nih.gov/pubmed/baseline/. I used the first 10 files as the data for this experiment.

Bi-LSTM CRF

For this model the input data needs to be updated so that all inputs are the same length, we apply padding at the end of the sentence of the length 50 and also apply this padding to the outputs. The input of the model then takes on the length of the padding and is pushed to an embedding layer. The embedding layer has the dimensions of the set of words $N + 1$. The embedding layer is then used as input to the bidirectional LSTM layer, which has 50 units and a recurrent dropout of 0.1. The next layer is the dense layer, with a ReLU activation that is then pushed to a CRF layer that is the size of the tags. The model uses an Adam optimizer combines the themes in RMSProp and stochastic gradient descent with momentum. The model was then trained on 22,307 samples and tested on 3,000 samples for 15 epochs. The model was then tested and validated on the validation and test set, both of which consisted of 8,263 samples.

Entity	Precision	Recall	F1 Score
disease	0.86	0.60	0.58
cellular-component	0.70	0.62	0.66
biological-process	0.66	0.56	0.61
molecular-function	0.77	0.74	0.75

Table 4.1: BiLSTM-CRF Model Performance during validation

LSTM with Character Embeddings

The next experiment is adapted from the experiments conducted by Lample et. al. It is very similar to the architecture in Figure 3.2, but instead of just using character embeddings, word embeddings are also used. The character embedding is generated by an LSTM model as described in Chapter 3. This model was trained on 20 epochs and used an Adam optimizer. Due to the complexity of the

Entity	Precision	Recall	F1 Score
disease	0.54	0.57	0.55
cellular-component	0.71	0.62	0.66
biological-process	0.57	0.54	0.55
molecular-function	0.76	0.73	0.75

Table 4.2: BiLSTM-CRF Model Performance during test

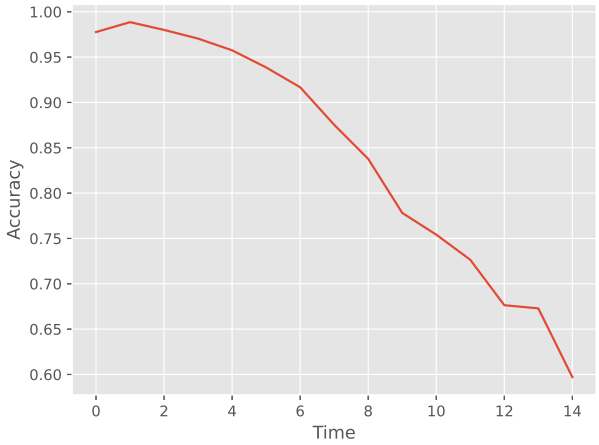

Figure 4.2: Accuracy during training of the gene model after tuning parameters and increasing the data

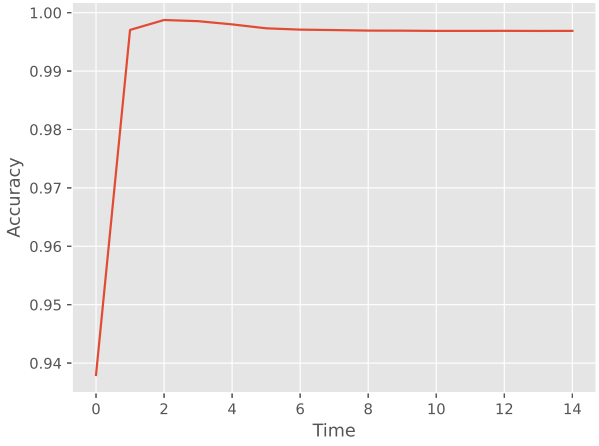

Figure 4.3: Accuracy during training of the disease model after tuning parameters and increasing the data

model more epochs were necessary to achieve a suitable score, the results show the results after training from 20 epochs using 35,000 samples and validated on 3,900 samples.

Figures 4.6 and 4.7 are somewhat similar to the first experiments and follow the trend of growth. Although this trend of growth could also be linked to model over-fitting as it was seen in the first two experiments. The results for the disease

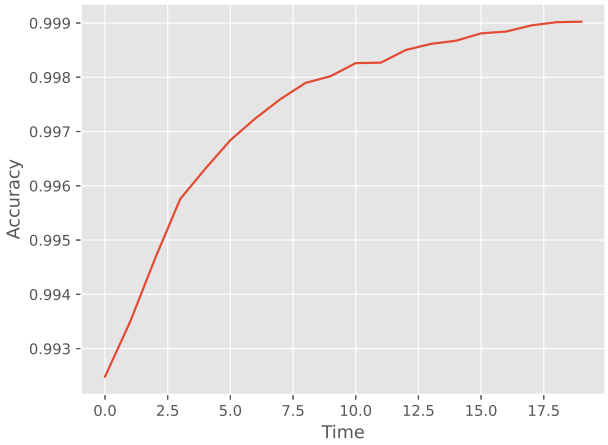

Figure 4.4: Accuracy during training of the disease model using character embeddings

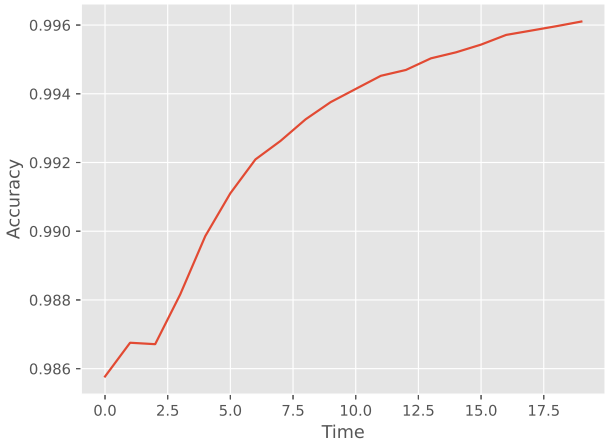

Figure 4.5: Accuracy during training of the gene model using character embeddings

model are slightly worse than the model with just simple word representations.

Entity	Precision	Recall	F1 Score
disease	0.71	0.66	0.69
cellular-component	0.65	0.69	0.67
biological-process	0.64	0.66	0.65
molecular-function	0.76	0.65	0.65

Table 4.3: BiLSTM-CRF with Character Embeddings Model Overall Performance

BiLSTM-CRF with ELMo Embeddings

Images 4.10 and 4.11 show the performance during testing for the BiLSTM model based on the architecture from flair. In total four experiments were run the standard ELMo word embeddings trained on domain-free text and one representation of ELMo trained on PubMed articles. The only difference between the two architecture types is the ELMo embeddings. Both experiments make use of Flair's stacked embeddings and use several word embeddings in combination. The first embedding is the GloVe embedding, the second a character embedding, and the last and ELMo embedding. The three embedding representations are concatenated and this represents a single vector after the *stacking*. The stacked embedding is then used as input to Flair's sequence tagger model. The model is a bidirectional LSTM model with a CRF layer that has a linear contains 256 hidden states, a dropout $p = 0.5$. The models were trained with the learning rate = .1, a mini-batch size of 32, and for 20 epochs. Flair recommends the following parameters except for the epochs for attaining state-of-the-art performance. The state-of-the-art performance was achieved with 150 epochs. The epochs are not fixed rather they represent a maximum variable and will be trained until the learning rate becomes too small. During the testing stage of the models, I learned that the model would train for 52 epochs on the data set of approximately $25,000$. This took over 14 hours on a GPU-enabled notebook to run, thus I tested with smaller epochs that attained similar performance. This is how I chose the end epoch size of 20 for all of the Flair models. This ran for about 8 hours on the GPU notebook and had good overall results, that were particularly promising during training. In figures 4.8 and 4.9 the scores during training for the disease model average in the 80% range and the upwards of 90% for the disease model. The accuracy is sometimes a peek into the actual performance but the model typically performs worse on the validation data. Overall the standard ELMo

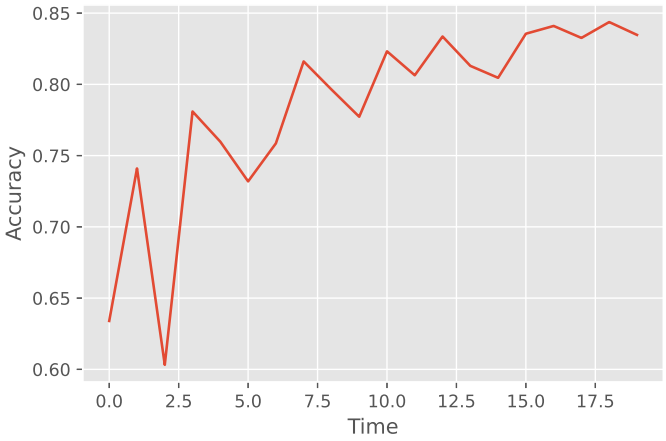

Figure 4.6: Accuracy during training of the gene model using the standard ELMo embeddings

model nonetheless outperforms the earlier models that only use word or character representations. For the disease class, the improvement is minimal, only 1%, on the other hand, the performance for cellular components and molecular

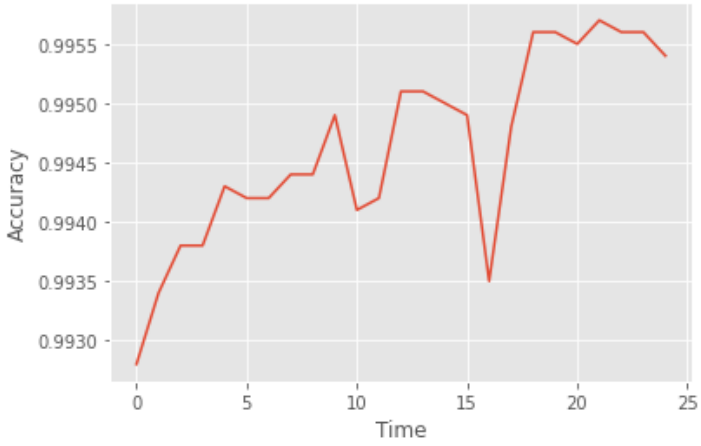

Figure 4.7: Accuracy during training of the disease model using the standard ELMo embeddings

Entity	Precision	Recall	F1 Score
disease	0.75	0.65	0.70
cellular-component	0.80	0.81	0.81
biological-process	0.84	0.75	0.79
molecular-function	0.77	0.92	0.83

Table 4.4: BiLSTM-CRF with ELMO (Standard) Embeddings Model Performance

function had a moderate boost in performance and a meager improvement for the biological processes. The second experiment is the ELMo model trained on PubMed articles. These embeddings represent domain-specific embeddings and my intuition is that these would outperform the standard ELMo representations. The "Bio"-ELMo embeddings do improve the F1 score of the model but also only slightly.

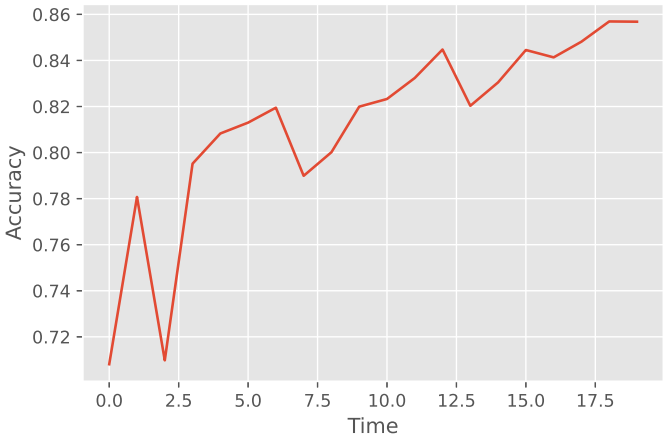

Figure 4.8: Accuracy during training of the gene model using biomedical ELMo embeddings

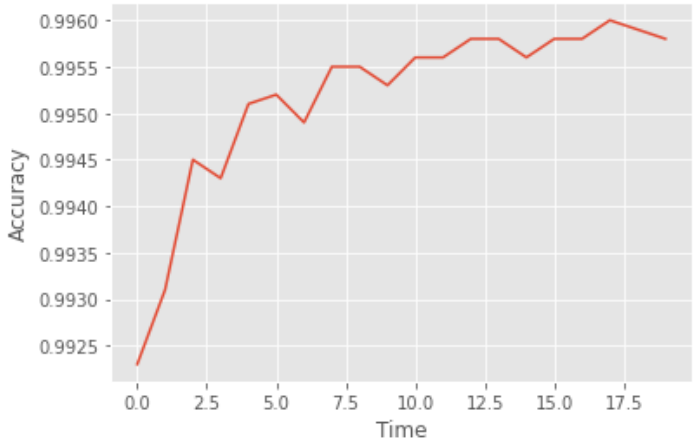

Figure 4.9: Accuracy during training of the disease model using biomedical ELMo embeddings

Entity	Precision	Recall	F1 Score
disease	0.73	0.73	0.73
cellular-component	0.76	0.90	0.82
biological-process	0.89	0.71	0.79
molecular-function	0.75	0.96	0.84

Table 4.5: BiLSTM-CRF with ELMO Biomedical Embeddings Model Performance

BiLSTM-CRF with BERT Embeddings

The next set of models have the same architecture of the above models and use BERT and BioBERT instead of the ELMo models. Much like the ELMo embeddings, the BioBERT embeddings are trained on PubMed abstracts. BioBERT was pre-trained on the PubMed dataset for 23 days and has surpassed the performance of biomedical named entity recognition at the time of its release of 62% (Lee et al. [2019]). The BERT model outperform the ELMo model in all classes. The same observations for the previous "Bio" embeddings are also present here, as the BioBERT model outperforms the BERT model. In the planning of this thesis, I theorized that the more complex a model's architecture is the better the performance of the model. It is with this assumption that I supposed the BioBERT model would outperform the rest of the models, although the the BERT that wasn't trained on PubMed outperformed it slightly. The assumption that domain-specific word embeddings outperform standard on the other hand remains true, albeit just a slight improvement. BERT word embeddings trained on PubMed was the best performing model and is with an F1 score in the 80th percentile it is quite dependable.

4.4.1 Performance on Gold Standard

This section documents the performance of disease models on the NCBI Disease validation data. Unfortunately I could there is not a corpus that represented the concept classes in the Gene Ontology, as some standards such as GENIA

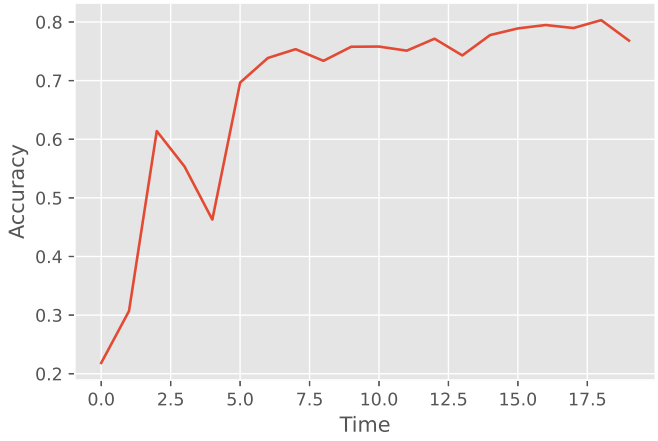

Figure 4.10: Accuracy during training of the gene model using the standard BERT base

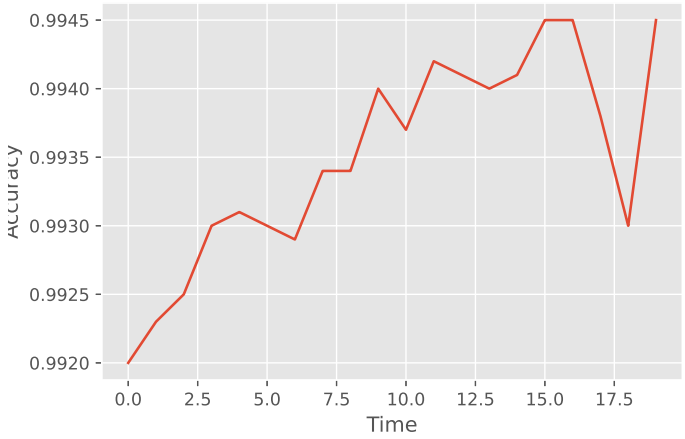

Figure 4.11: Accuracy during training of the disease model using the standard BERT base

Entity	Precision	Recall	F1 Score
disease	0.78	0.65	0.71
cellular-component	0.91	0.82	0.87
biological-process	0.95	0.71	0.82
molecular-function	0.89	0.89	0.89

Table 4.6: BiLSTM-CRF with BERT (base) Embeddings Model Performance

and focused on the sub-entities, but not the processes and functions the ontology focuses on. In the below table we can see that in the models there is generally a high precision and a very low recall. This shows that the model returns very few results, but the results they do produce are mostly correct.

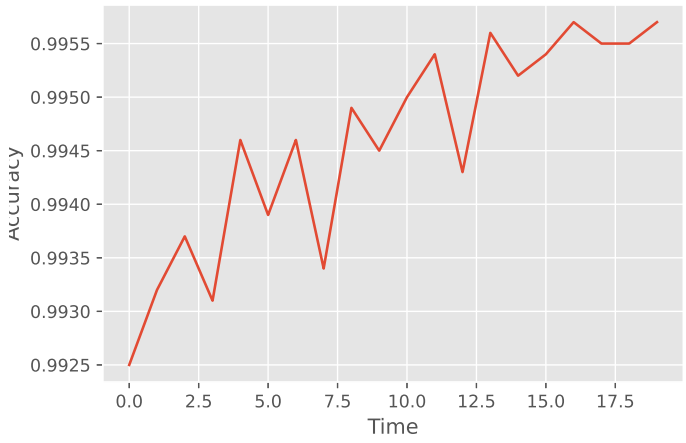

Figure 4.12: Accuracy during training of the disease model using the BioBERT

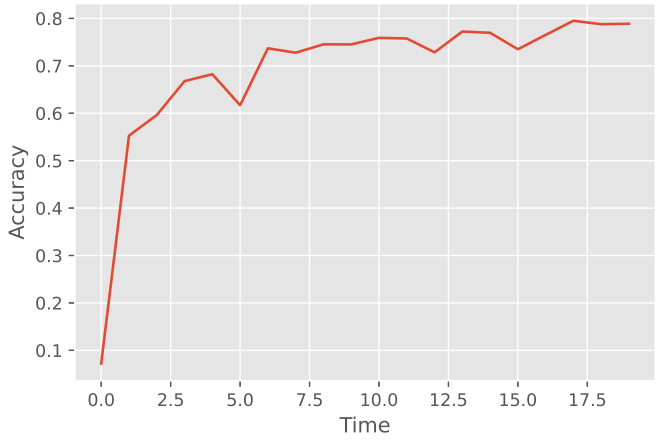

Figure 4.13: Accuracy during training of the gene model using the BioBERT

Entity	Precision	Recall	F1 Score
disease	0.76	0.59	0.66
cellular-component	0.86	0.90	0.88
biological-process	0.87	0.73	0.80
molecular-function	0.80	0.94	0.86

Table 4.7: BiLSTM-CRF with BioBERT Embeddings Model Performance

4.5 Conclusions

The goal of this thesis was to examine the current approaches to named entity recognition of the biomedical domain. In this thesis, I explored the architectures of named entity systems and their components, namely word embeddings. In the introduction I outlined three main goals of this thesis:

- examine the neural network approaches to named entity recognition using recurrent networks

Model	Precision	Recall	F1 Score
disease(BERT)	0.91	0.21	0.35
disease(BioBERT)	1.00	0.36	0.53
disease(ELMo)	0.98	0.18	0.31
disease(BioELMo)	1.00	0.26	0.42
disease(LSTMChar)	0.87	0.17	0.28
disease(LSTMCRF)	0.82	0.14	0.23

Table 4.8: Model Performance on NCBI Disease Validation Data

- examine the influence of the performance of these models with different word embeddings
- investigate the effects of word embeddings on ambiguous text.

Chapters 2 and 3 focus on the first goal of this thesis and are an in-depth examination of named entity recognition approaches. The experiments in this chapter conclude that different word embeddings do have an influence on the model performance and that on this specific data set with the set parameters ELMo is the best performing when combined with a Bidirectional LSTM with a CRF layer.

The third goal which was identified in the official thesis statement was unfortunately not met. There are two main reasons for this, one is the manner of the ambiguity of the entities that I focused on for this thesis. During the topic exploration, I chose the ambitious goal of focusing on ambiguity in the text as it is a known "pain point" in the industry, and many approaches to ambiguity are trivial and can revert to using a simple white list of terms to avoid for the introduction of false positives. I aimed to find out how the approaches could be applied to biomedical texts, assuming that much like ambiguous text of other domains were similar to the ambiguous texts of this domain. While there is no shortage of ambiguity in the biomedical domain the *type* of the ambiguity is what makes this task extremely difficult to test. The entities of interest for this thesis are of two types: the genetic domain and the bio-processes that interact with genes, and diseases. In the genetic domain, the ambiguity is not entirely polysemous, in other words, a representation for the gene NAP1, this one gene representation can be mapped to 5 different genetic structures (Stevenson and Guo [2010]). This kind of ambiguity would not be captured by this system as it would recognize all of them in theory, the ambiguity is not the question of identifying it as a gene class but rather mapping it to a specific gene using some standard identifiers. The experiments I conducted only tag the data as one of the classes in the ontology, it does not identify which entity it is using some unique identifier. The disease on the other hand does have ambiguous text in the sense that in this context the word is or is not a given disease two examples mentioned earlier were *AIDS*/aids and *cold*/cold. These are two that are mentioned often in literature when discussing ambiguity in the biomedical domain. I attempted to come up with a longer list of such examples for diseases and found that very few diseases have this type of ambiguity. This made it difficult to systematically test the disambiguation of the text as with so few results it only sheds light on this specific type of small case disambiguation. I have examined on the small scale the performance of contextualized word embeddings on the two cases. All

of the models do not detect the difference between the two colds, and do not match AIDS at all. Cold is identified in the ontologies and during the tagging this introduced both forms as entities (negative and positive hits), as the tagging doesn't discriminate based on context. The small scale evaluation does not provide concrete conclusions about the performance of the contextualized word embeddings on ambiguity, and this needs to be explored more in depth.

This thesis can be used as a survey to the approaches of named entity recognition using neural networks. This thesis explored the the theme of biomedical named entity recognition, and found that tagging data sets with a dictionary yields to unsatisfactory performance on neural networks. While the performance of the models where hopeful on the custom data sets they performed very poorly on the standard test data, and are unable to capture many entities.

4.5.1 Future and Related Work

At the time of writing this thesis new language model representations were being developed and would be useful extensions to the experiments conducted during this thesis. ELECTRA, T5, and GPT3 are a few of the newer models which build off of BERT and transfer learning which would make promising candidates for improving the performance. I would also like to continue the work on ambiguity, but would need a more reliable set of data to create models for this experiment. This work was mainly an exploration to see if more semi-automated approaches can be used as data for a neural network. The finding show poor performance, and displays the need for appropriate training data. The work on this topic is continual and as the members of the NLP community continue to make breakthroughs with the help of GPUs and TPUs we will see improvements in the field.

Bibliography

Aho and Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, pages 333–340, 1975.

Akbik, Bergmann, Blythe, Rasul, Schweter, and Vollgraf. Flair: An easy-to-use framework for state-of-the-art nlp. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, 2019.

Ananiadou, Kell, and Tsujii. Text mining and its potential applications in systems biology. *Trends in Biotechnology*, pages 571–579, 2006.

Appelt, Hobbs, Bear, Israel, and Tyson. Fastus: A finite-state processor for information extraction from real-world text. *In Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1172–1178, 1993.

Ashburner, Ball, Blake, Botstein, Butler, and Cherry. Gene ontology: Tool for the unification of biology. *The Gene Ontology Consortium. Nat Genet*, pages 25–29, 2000.

Bahdanau, Cho, and Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, pages 1–15, 2014.

Baum and Petrie. Statistical inference for probabilistic functions of finite state markov chains. *Annals of Mathematical Statistics*, pages 1554–1563, 1966.

Bikel, Schwartz, and Weischedel. An algorithm that learns what's in a name. *Machine learning*, pages 211–231, 1999.

Blaschke, Hirschman, and Valencia. Information extraction in molecular biology. *Briefings in Bioinformatics*, pages 154–165, 2002.

Olivier Bodenreider. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, pages D267–D270, 2004.

Canale, Lisena, and Troncy. A novel ensemble method for named entity recognition and disambiguation based on neural network. In *International Semantic Web Conference*, pages 91–107, 2018.

Cetoli, Akbari, Bragaglia, O'Harney, and Sloan. Named entity disambiguation using deep learning on graphs. *Advances in Information Retrieval*, pages 78–86, 2018.

Chinchor and Sundheim. Muc-5 evaluation metrics. In *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*, 1993.

Cimiano, Hartung, and Ratsch. Finding the appropriate generalization level for binary ontological relations extracted from the genia corpus. pages 191–196, 2006.

The Gene Ontology Consortium. The Gene Ontology resource: enriching a GOld mine. *Nucleic Acids Research*, pages D325–D334, 2020.

Crichton, Pyysalo, Chiu, and Korhonen. A neural network multi-task learning approach to biomedical named entity recognition. pages 1–15, 2017.

Dale. Symbolic approaches to natural language processing. pages 1–9, 2000.

Deerwester, Dumais, Furnas, Landauer, and Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, pages 391–407, 1990.

Devlin, Chang, Lee, and Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. pages 1–16, 2018.

Dogan, Leaman, and lu. Ncbi disease corpus: A resource for disease name recognition and concept normalization. *Journal of biomedical informatics*, pages 1–11, 2014.

Duranti and Goodwin. *Rethinking context: Language as an interactive phenomenon*, volume 11. 1992.

Egorov, Yuryev, and Daraselia. A Simple and Practical Dictionary-based Approach for Identification of Proteins in Medline Abstracts. *Journal of the American Medical Informatics Association*, pages 174–178, 2004.

Ethayarajh. How contextual are contextualized word representations? comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 55–65, 2019.

Friedman and Elhadad. Natural language processing in health care and biomedicine. In *Biomedical informatics*, pages 255–284. 2014.

Gage. A new algorithm for data compression. pages 23–38, 1994.

Halliday and Matthiessen. *Halliday's introduction to functional grammar*. Routledge, 2013.

Hirschman, Morgan, and Yeh. Rutabaga by any other name: extracting biological names. *Journal of Biomedical Informatics*, pages 247–259, 2002.

Hochreiter and Schmidhuber. Long Short-Term Memory. *Neural Computation*, pages 1735–1780, 1997a.

Hochreiter and Schmidhuber. Long short-term memory. *Neural computation*, pages 1735–1780, 1997b.

Daniel Jurafsky and James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, volume 2. 2008.

Kim, Ohta, Tateisi, and Tsujii. GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics*, pages i180–i182, 2003.

Kipf and Welling. Semi-supervised classification with graph convolutional networks. pages 1–14, 2016.

Krifka. Basic notions of information structure. *Acta Linguistica Hungarica*, pages 243–276, 2008.

Lample, Ballesteros, Subramanian, Kawakami, and Dyer. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, 2016.

Lee, Yoon, Kim, Kim, Kim, So, and Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, pages 1234–1240, 2019.

Liddy. Natural language processing. 2001.

Locke and Bogin. Language and life history: a new perspective on the development and evolution of human language. *The Behavioral and brain sciences*, pages 259–80; discussion 280–325, 2006.

Makino, Ohta, Tsujii, et al. Tuning support vector machines for biomedical named entity recognition. In *Proceedings of the ACL-02 workshop on Natural language processing in the biomedical domain*, pages 1–8, 2002.

Mikolov, Chen, Corrado, and Dean. Efficient estimation of word representations in vector space. pages 1–12, 2013.

Behrang Mohit. *Named Entity Recognition*, pages 221–245. 2014.

Nadeau and Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, pages 3–26, 2007a.

Nadeau and Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, pages 3–26, 2007b.

Christopher Olah. Understanding lstm networks, 2015. URL `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Pascanu, Mikolov, and Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

Pennington, Socher, and Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

Peters, Neumann, Iyyer, Gardner, Clark, Lee, and Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, 2018.

Pyysalo, Ginter, Heimonen, Björne, Boberg, Järvinen, and Salakoski. Bioinfer: A corpus for information extraction in the biomedical domain. *BMC bioinformatics*, page 50, 2007.

Rau. Extracting company names from text. In *[1991] Proceedings. The Seventh IEEE Conference on Artificial Intelligence Application*, pages 29–32, 1991.

Schriml, Mitraka, Munro, Tauber, Schor, Nickle, Felix, Jeng, Bearer, Lichenstein, Bisordi, Campion, Hyman, Kurland, Oates, Kibbey, Sreekumar, Le, Giglio, and Greene. Human Disease Ontology 2018 update: classification, content and workflow expansion. *Nucleic Acids Research*, pages D955–D962, 2018.

Selman. Connectionist systems for natural language understanding. *Artificial Intelligence Review*, pages 23–31, 1989.

Sennrich, Haddow, and Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016.

Smith, Ashburner, Rosse, Bard, Bug, Ceusters, Goldberg, Eilbeck, Ireland, Mungall, et al. The obo foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, pages 1251–1255, 2007.

Stevenson and Guo. Disambiguation in the biomedical domain: the role of ambiguity type. *Journal of Biomedical Informatics*, pages 972–981, 2010.

Sun, Lin, Tang, Yang, Ji, and Wang. Modeling mention, context and entity with neural networks for entity disambiguation. In *Proceedings of the 24th International Conference on Artificial Intelligence*, page 1333–1339, 2015.

Charles Sutton, Andrew McCallum, et al. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, pages 267–373, 2012.

Swanson. Fish oil, raynaud's syndrome, and undiscovered public knowledge. *Perspectives in Biology and Medicine*, pages 7–18, 1986a.

Swanson. Undiscovered public knowledge. *The Library Quarterly*, pages 103–118, 1986b.

Swanson. Migraine and magnesium: Eleven neglected connections. *Perspectives in Biology and Medicine*, pages 526 – 557, 1988.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. pages 1–11, 2017.

Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, pages 260–269, 1967.

Wang, Xiao, Lin, and Zhang. Efficient approximate entity extraction with edit distance constraints. pages 759–770, 2009.

Wheeler, Barrett, Benson, Bryant, Canese, Chetvernin, Church, DiCuccio, Edgar, Federhen, et al. Database resources of the national center for biotechnology information. *Nucleic acids research*, pages D13–D21, 2007.

Wilkins and Wakefield. Brains evolution and neurolinguistic preconditions. *Behavioral and Brain Sciences*, page 161–182, 1995.

Yamada, Shindo, Takeda, and Takefuji. Joint learning of the embedding of words and entities for named entity disambiguation. *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 1–10, 2016.

Zhai, Nguyen, and Verspoor. Comparing cnn and lstm character-level embeddings in bilstm-crf models for chemical and disease named entity recognition. pages 38–43, 2018.

Zhang and LeCun. Text understanding from scratch. pages 1–10, 2015.

Zucco, Calabrese, and Cannataro. Sentiment analysis and affective computing for depression monitoring. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1988–1995, 2017.

List of Figures

1.1 Language stratification, adapted from Halliday and Matthiessen [2013] 4

2.1 Sample structure for HMM NER Model, adapted from (Bikel et al. [1999]), Figure 3 11
2.2 Architecture of recurrent neural network, adapted from Olah [2015] 13
2.3 LSTM Cell, adapted from Olah [2015] 14
2.4 Architecture of the network, adapted from Lample et al. [2016], where l_i and r_i represent left and right contexts, and the contextual representation c_i is produced from the concatenation. This figure used the IOB tagging schema. 15

3.1 Architecture of Continuous Bag of Words network, adapted from Mikolov et al. [2013] Figure 1. 23
3.2 Architecture of BiLSTM with character embeddings, adapted from Lample et al. [2016] 28
3.3 Architecture of ELMo 31
3.4 Transformer Architecture (Vaswani et al. [2017], Figure 1) 32
3.5 Scaled Dot Product Attention and Multi-Head Attention (Vaswani et al. [2017], Figure 2) 33
3.6 BERT Architecture (Devlin et al. [2018], Figure 1) 33

4.1 Sample Input of formatted sentences 41
4.2 Accuracy during training of the gene model after tuning parameters and increasing the data 43
4.3 Accuracy during training of the disease model after tuning parameters and increasing the data 43
4.4 Accuracy during training of the disease model using character embeddings 44
4.5 Accuracy during training of the gene model using character embeddings 44
4.6 Accuracy during training of the gene model using the standard ELMo embeddings 45
4.7 Accuracy during training of the disease model using the standard ELMo embeddings 46
4.8 Accuracy during training of the gene model using biomedical ELMo embeddings 46
4.9 Accuracy during training of the disease model using biomedical ELMo embeddings 47
4.10 Accuracy during training of the gene model using the standard BERT base 48
4.11 Accuracy during training of the disease model using the standard BERT base 48
4.12 Accuracy during training of the disease model using the BioBERT 49
4.13 Accuracy during training of the gene model using the BioBERT . 49

List of Tables

4.1 BiLSTM-CRF Model Performance during validation 42
4.2 BiLSTM-CRF Model Performance during test 43
4.3 BiLSTM-CRF with Character Embeddings Model Overall Performance 44
4.4 BiLSTM-CRF with ELMO (Standard) Embeddings Model Performance 46
4.5 BiLSTM-CRF with ELMO Biomedical Embeddings Model Performance 47
4.6 BiLSTM-CRF with BERT (base) Embeddings Model Performance 48
4.7 BiLSTM-CRF with BioBERT Embeddings Model Performance . . 49
4.8 Model Performance on NCBI Disease Validation Data 50

List of Abbreviations

A. Attachments

A.1 First Attachment