Norwegian University
of Life Sciences

Master's Thesis 2021 30 ECTS
Faculty of Science and Technology

Comparative study of NER using Bi-LSTM-CRF with different word vectorization techniques on DNB documents

Meera Joseph
Master of Science in Data Science

Preface

This thesis marks the end of my two-years master's course in Data science. This is written at the Faculty of Science and Technology at the Norwegian University of Life Sciences (NMBU) in 2021. This thesis has been carried out in collaboration with the IT Emerging Technologies at the DNB bank.

Firstly, I would like to thank Jan Thomas Lerstein, head of IT-Emerging Technologies DNB, for giving me this opportunity to carry out my thesis at DNB. I would also like to thank my supervisors Kristian Hovde Liland and Oliver Tomic, Faculty of Science and Technology (REALTEK) NMBU, for their excellent guidance, support and feedback throughout the process. Furthermore, I'd like to thank my colleagues at DNB, particularly Mateo Caycedo Alvarez, Abhilash Nair and Kamal Singh, for their support and motivation.

I would also like to acknowledge the contributions of my family in helping me and supporting me to complete my thesis successfully during the Covid times.

Ås, 1st June, 2021

_____

Meera Joseph

Abstract

The presence of huge volumes of unstructured data in the form of pdf documents poses a challenge to the organizations trying to extract valuable information from it. In this thesis, we try to solve this problem as per the requirement of DNB by building an automatic information extraction system to get only the key information in which the company is interested in from the pdf documents. This is achieved by comparing the performance of named entity recognition models for automatic text extraction, built using Bi-directional Long Short Term Memory (Bi-LSTM) with a Conditional Random Field (CRF) in combination with three variations of word vectorization techniques. The word vectorisation techniques compared in this thesis include randomly generated word embeddings by the Keras embedding layer, pre-trained static word embeddings focusing on *100-dimensional* GloVe embeddings and, finally, deep-contextual ELMo word embeddings. Comparison of these models helps us identify the advantages and disadvantages of using different word embeddings by analysing their effect on NER performance. This study was performed on a DNB provided data set. The comparative study showed that the NER systems built using Bi-LSTM-CRF with GloVe embeddings gave the best results with a micro F1 score of 0.868 and a macro-F1 score of 0.872 on unseen data, in comparison to a Bi-LSTM-CRF based NER using Keras embedding layer and ELMo embeddings which gave micro F1 scores of 0.858 and 0.796 and macro F1 scores of 0.848 and 0.776 respectively. The result is in contrary to our assumption that NER using deep contextualised word embeddings show better performance when compared to NER using other word embeddings. We proposed that this contradicting performance is due to the high dimensionality, and we analysed it by using a lower-dimensional word embedding. It was found that using *50-dimensional* GloVe embeddings instead of *100-dimensional* GloVe embeddings resulted in an improvement of the overall micro and macro F1 score from 0.87 to 0.88. Additionally, optimising the best model, which was the Bi-LSTM-CRF using *100-dimensional* GloVe embeddings, by tuning in a small hyperparameter search space did not result in any improvement from the present micro F1 score of 0.87 and macro F1 score of 0.87.

Contents

List of figures

List of Tables

Chapter 1 Introduction

1.1 Background

In a study conducted by the International Data Corporation (IDC) to measure the amount of digital data between 2005 to 2020, it was predicted that unstructured data, e.g., text files, e-mails, presentations etc. would account for *95%* of the global data with an estimated increase of *65%* per year. IBM also conducted a similar study, and it was found that *2500,000* Terabytes of data are produced daily [1, 2]. Additionally, IDC's market survey between 2009 and 2020 suggests that the rate at which digital data is growing is not proportionally compensated by staffing and investment to manage it [2, 3].

Advancement in the field of information and communication technology has also led to the generation of more data, in the form of data created and shared. This trend in the growth of data associated with digitalization has triggered development in the field of automated information extraction systems for deriving value from large scale unstructured data.

Digital data is divided into three categories, namely structured, unstructured, and semi-structured data. Firstly, structured data also referred to as quantitative data, has an organized structure and clearly defined data types. They have a predefined format and can be stored and queried in a relational database, thus making them easy to export, searchable by data type, and easy to organize. Relational databases, electronic spreadsheets, etc., are a few examples of structured data. Secondly, unstructured data, otherwise known as qualitative data, as they mostly contain factual information that is not measurable, refers to text-heavy data with multiple data types like dates, names, numbers, and images, thus making them ambiguous and difficult to search. They do not have a rigid structure and cannot be stored in tables. PowerPoint, word documents, email, PDF, etc., are a few of the examples in this category. Finally, we have semi-structured data, which is not as disordered as unstructured data, but do not have a rigid and pre-defined standard like the structured data. It can have information from multiple data sources. An example of semi-structured data is JSON documents, where the data

does not have fixed fields associated with it and cannot be stored in a table without transforming it to a structured form [4-6].

This thesis focusses on unstructured data and therefore this will be further discussed to understand them in terms of the volumes in which they are generated and their ability to give rise to useful information through data analytics.

Unstructured data can take different forms. It can be human-generated in the form of text files like emails, presentations, excel files or as data generated in social media like the MP3, digital photos, audio recordings, video files and so on. Unstructured data can also be machine-generated like satellite imagery, sensor data, traffic, temperature, etc [4]. Thus, it can be said that unstructured data occur in *large volumes* and are *complex* due to this diversity in the sources and formats [1].

Studies [1, 6, 7] show that unstructured data play a significant role in gaining a deep insight into many factors that influence an organisation's functioning and growth like business trends, competition in the field, etc. It also enables an organisation to improve productivity by gathering and analysing customer trends and interests [6]. The presence of huge volumes of offline data in the form of applications, letters, papers, documents, certificates, surveys, agreements, etc. should also be considered while analysing the sources of unstructured data. These large volumes of complex unstructured data make the process of information extraction quite challenging. Historically, vital information from such documents have been extracted by manual intervention. But this process is highly time-consuming, labour-intensive, and inefficient. Inefficiency can be due to the fact that manual interventions are subject to human errors and subjectivity in making decisions. Irrespective of the amount manual labour invested in assessing unstructured data, it becomes difficult to catch up when the volumes increase swiftly, hence it cannot be considered as a feasible solution for information extraction [8]. This is where the ability to extract keywords from documents and classify them into predefined categories would serve highly beneficial. This process of information extraction is referred to as Named Entity Recognition (NER) in the data science world and will be further discussed in Sec. 3.1 [9].

For this thesis, we are dealing with unstructured data in the banking sector. A brief understanding of its presence in the financial industry will help us recognize the relevance of implementing an automated information extraction system specific to the field. Huge volumes of data related to customers, business, processes, and employee engagement are generated in the financial sector. These are in the form of banking transactions, payment orders, account statements, online forms, chatbot logs, loan documents, emails, text messages, audio/video communications etc. The exploding rate of growth of data comes with its challenges concerning storage and management, but it holds opportunities for growth in the sector. Performing analytics on this data may give us insight into the banking business, customer retention, customer behaviour, decision making, managing risk, maintaining customer relationships, and performance management process of a bank. It can also make the bank capable of offering a superior banking experience based on the analysis conducted. This helps in making banks equipped for competing with their counterparts and in offering what they promise efficiently [7, 10].

In general, the process of information extraction comes with many pros and cons and some of them has already been discussed in detail in the above sections. The positive impact of automating information extraction process with regards to time and cost can be seen through an example of JP Morgan chase & co, which is one of the largest banks in the USA. They deployed a program called Contract Intelligence (COiN) for automating the process of document review based on unsupervised machine learning. This task was previously performed by lawyers and spent about *360,000* man hours each year on that task. In a [11] case study conducted on COiN, it was found that about *13%* of the effort spent by the lawyers were saved in the process. The program can review *12,000* credit agreements a year. The *360,000* man hours spent on the process a year was cut down by reducing the time spent on extracting information from a single credit agreement to a few seconds through this implementation. The amount of money spent by the company on employing people for performing the task was also reduced. In addition to its positive impact on cost and time, automation of the process has proved to achieve higher accuracy in document review in comparison with the possible manual errors that could occur as the decisions were not prejudiced or influenced by emotions or efficiency of the person [11].

However, there are some drawbacks associated with automating information extraction. Extracting key information from the full data could be valuable in terms of insights but the

extracted information might be sensitive, and hence it would risk the privacy of the individual or business involved. This led to the development of privacy preserving data mining (PPDM) which is a subfield of data mining which considers sensitive information where steps are taken to safeguard the misuse of this information [12]. It should also be noted that though automating NER removes prejudiced human decisions, it can accidentally extract wrong values and misclassify them. For example: consider an automated information extraction system which acts as a downstream task for another application that stores the address of a person privately and displays the name of a person in a way it is visible to the public. Automation error could lead to the wrong classification of a person's address in the name column, thus making it publicly visible and endangering the privacy of a person. Additionally, merging data sources to create a single source for building an automatic NER comes with legal risks associated with it, in terms of data sharing and manipulation.

1.2 Problem statement

1.2.1 What is the problem we are trying to solve?

This thesis is being done as a prototype for a solution at the DNB IT emerging technologies team. DNB ASA is one of Norway's largest financial services group. The DNB IT emerging technologies team strives to develop novel solutions to technologically enhance the banking system. Following a request from an internal team, the IT emerging technologies were interested in developing a system for extracting key elements from unstructured textual documents published in the *Oslo stock exchange* website. These are letters sent to shareholders for attending general meetings organized by the company selling the shares. The key elements of interest for them are company name, meeting date, meeting address and company deadline. The details about the data set are further explained in the Sec. 2.1. In this process, DNB acts as an intermediary between the company and the shareholders. The focus of this thesis is to build an automatic information extraction system based on Named Entity Recognition (NER), to reduce the time and effort spent on it, using different deep learning methods.

1.2.2 Why is solving this problem relevant?

Based on the discussions I have had with DNB, there exist huge volumes of unstructured data in the form of PDF's, emails, forms, etc. which are stored in the cloud and this data can be used by the bank for its functional development. For this thesis, unstructured data in the

form of pdf documents are used. These documents are processed and converted into structured forms for performing information extraction. Moreover, the data extracted by the NER tasks can be used as an input to other downstream tasks in a value chain. One such pipeline is where the information extracted would be sent to the shareholders who have subscribed to this service at DNB. Presently this is achieved manually with the help of an analyst who is continuously sifting through the meeting documents published by the *Oslo stock exchange*, looking for relevant and useful information from them.

Larger scope of NER at DNB include:

1. Extracting relevant information or keywords from business agreements between DNB and other third parties, for example, mortgage applications making the process smoother and faster in effect.
2. Identifying threats to the bank's IT systems by processing documents and emails. This can be done by extracting certain types of words from the email messages and then classifying them as a threat or not.
3. Processing and categorising transactions in areas like private banking and wealth management.

The presence of large volumes of unstructured data in both the banking industry and outside as discussed in the background section extends the relevance of this thesis to broader fields of application, for example, first-level filtering of candidates based on their resumes by extracting only the entities of interest like name, education, years of experience, etc. without spending hours and reading thousands of resumes.

1.2.3 How are we trying to solve the problem?

We aim to solve the problem by achieving two major goals and also by reaching a conclusion concerning a hypothesis put forward based on previous research in the field.

Through this study we aim to achieve two major goals:

1. Comparative study of deep learning models based on their performance in Named Entity Recognition task (NER) in extracting entities specific to DNB documents. The deep learning models are based on Bidirectional Long Short-Term Memory (Bi-LSTM) with Conditional Random Fields (CRF) in combination with different word vectorization techniques.

2. Build an automatic NER system with good performance that works well with these specific documents through hyperparameter tuning.

As a subgoal, due to the lack of annotated data for the shareholder's general meeting invite documents for training the models, we manually create the data set in this thesis (see Sec. 2.3).

Hypothesis statement:

1. Using a deep contextualized word embeddings would result in better performance of NER system [13].

1.2.4 Why did we choose this method for solving the problem?

Training of large neural networks has become quite popular recently with an increasing amount of data, and the availability of cheap storage and computational power. The research focuses on building NER using Bi-LSTM with CRF. Based on the information collected on the previous researches done in the field [14-17] among various deep learning approaches used to solve the information extraction problem, methods using Bi-LSTM with a top-level CRF model have been shown to achieve the good results. Therefore, using this architecture as the base model for building a NER model for information extraction in this paper.

1.3 Related work

Named Entity Recognition using deep learning for information extraction is an area of extensive research and therefore the number of research papers published in this topic is vast. This section should not be considered as an exhaustive overview of all the papers in the field, but a summary of research performed in the field based on a few selected papers. More related works are discussed in Chapter 4 to explore the scope for future work.

In a paper by Ismail et al. [18], NER is performed on data in the Arabic language using deep learning approaches. They propose an approach based on deep learning to reduce the dependency on external resources and hand-made feature engineering. A Bi-LSTM-CRF was used in combination with character representation of words and pre-trained word embeddings. Fast Text [19] is used as the pre-trained word embedding here. Arabic being rich in morphological forms make the vocabulary sizes larger and out of vocabulary rate relatively

higher. It also highlights that for languages like English without morphological richness, the use of character representation is not mandatory to get the best results. But few other researches such as [11, 12, 14, 15] showed that using character embeddings along with word embeddings results in an improvement in the model performance on CoNLL 2003 data set [20]. Peters et al. [13] proposed that deep contextualized word embeddings from language model known as Embedding from Language Model (ELMo) resulted in improvement in performance of a NER model based on Bi-LSTM-CRF using random embeddings on CoNLL 2003 data set.

In a study by Huang et al. [17] it was proposed that Bi-LSTM-CRF is robust and its performance is less dependent on the type of word embedding used. It uses Bi-LSTM to capture the contextual information and a CRF top layer is used to compare the sentence-level tag information before making the final predictions regarding the label. On the other hand, in a study by Hovy et al. [14] various word embeddings with Bi-LSTM NER in combination with character embeddings were compared on CoNLL 2003 data set and it was found that *100-dimensional* GloVe embeddings resulted in the best performance in comparison to other pre-trained word embeddings used.

Zhai et al. [21] proposed the concept of using pre-trained word embeddings trained on domain specific corpus. A model called EBC-CRF with domain specific pre-trained word embeddings was proposed and it was found to improve the performance of chemical NER on BioSematics patent corpus and Reaxys gold set [22]. EBC-CRF is a combination of Bi-LSTM-CNN-CRF with ELMo contextual word embeddings and pre-trained word embeddings. In our study, we compare the performance of Bi-LSTM-CRF based NER when they are coupled with different types of word vectorisation techniques. A study by Zhu et al. [23] showed that using clinical pre-trained ELMo word embeddings improved the performance of Bi-LSTM-CRF NER on 2020 i2b2/VA data set [24].

1.4 Structure of the thesis

The thesis begins with the Chapter 1 introduction section, which holds information about why the thesis is relevant , what is the problem we are trying to solve through this study and how are we trying to solve it with the help of a goals and hypothesis. It also explains why we choose the methods we used and briefly discusses previous work done by researchers in the field.

Chapter 1 is followed by the materials section in Chapter 2. It contains information about the data set on which the methods are applied which includes the process of building the data set starting from the information extraction from PDF documents to converting it to suitable form that can be given as input to the Bi-LSTM-CRF model. Chapter 3, methods section, discusses the steps followed to perform the experiments. It also includes the theory behind different components used in the methods section for performing the comparative study of the NER models using deep learning methods based on Bi-LSTM in combination with CRF. It also includes a brief introduction into the concept of Named Entity Recognition, followed by the theory behind different word vectorisation techniques and the tagging scheme used for preparing the data set as described in Chapter 2. In Chapter 4, the results of the deep learning models are reported, along with the results of the data preparation step. The best results of the models are also tabulated and discussed along with results from previous researches for comparison. The results of the experiments are and findings based on the hypothesis are summarised as a conclusion in Chapter 5.

Chapter 2 Materials

This section gives an overview of the data set used in the thesis along with tagging scheme used to understand the dataset better. It also discusses the steps taken to manually build the data set and the theory behind various components involved.

2.1 About the data set

To date, annotated data for this specific task does not exist. Hence we are building the evaluation corpus manually. The process of building the data set is explained in Sec. 2.3. The PDF documents used in this study are notices sent out by companies inviting their shareholders to participate in the general meeting. The Oslo stock exchange publishes these meeting invites, and DNB regularly stores these in the AWS cloud storage. Due to time constraints we selected only a few documents out of the total available for the thesis since the data set was created manually, as described in Sec. 2.3. It was found through experiments that while training the model with this limited set of documents, the model gave a good generalisation of behaviour and was able to learn features and make accurate predictions on unseen data. The general meetings are organised annually between the company and the shareholders. In this process, DNB acts as an intermediary for helping the exchange of information between them. The complete data set contains *133* documents, with a total of *747* sentences. After manual annotation, which involves the process of reading through the text to identify key words and assign them to corresponding class labels, the data set contained a total of *31721* words. The process of manual annotation is explained in detail in Sec. 2.3.5.

The words or group of words that we are interested in extracting from the text are referred to as named entities. In the data set each word was assigned with a tag corresponding to the Inside outside beginning (IOB2) tagging scheme [25, 26] along with the class label (see Sec. 2.2). Figure 2.1 shows the distribution of entities belonging to each type of tag except the "*O*" tag, and it shows that the majority of entities belong to tag *25 - company name*. Figure 2.2 shows that out of a total of *31721* words *4691* words are named entities.

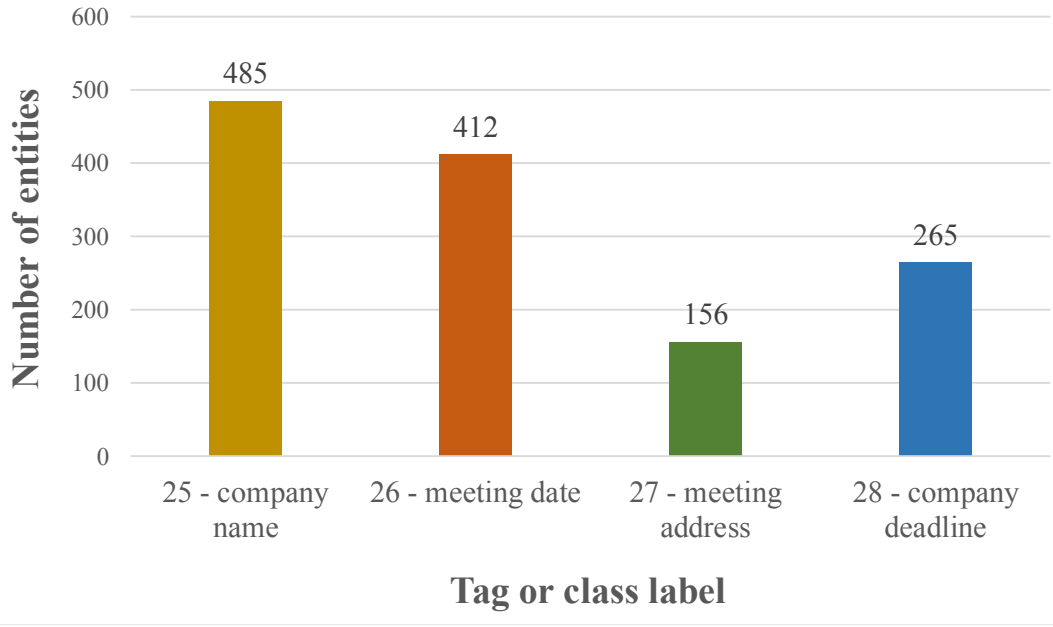


Figure 2.1:Distribution of tags in the data set without including the "O" tag [2.2].

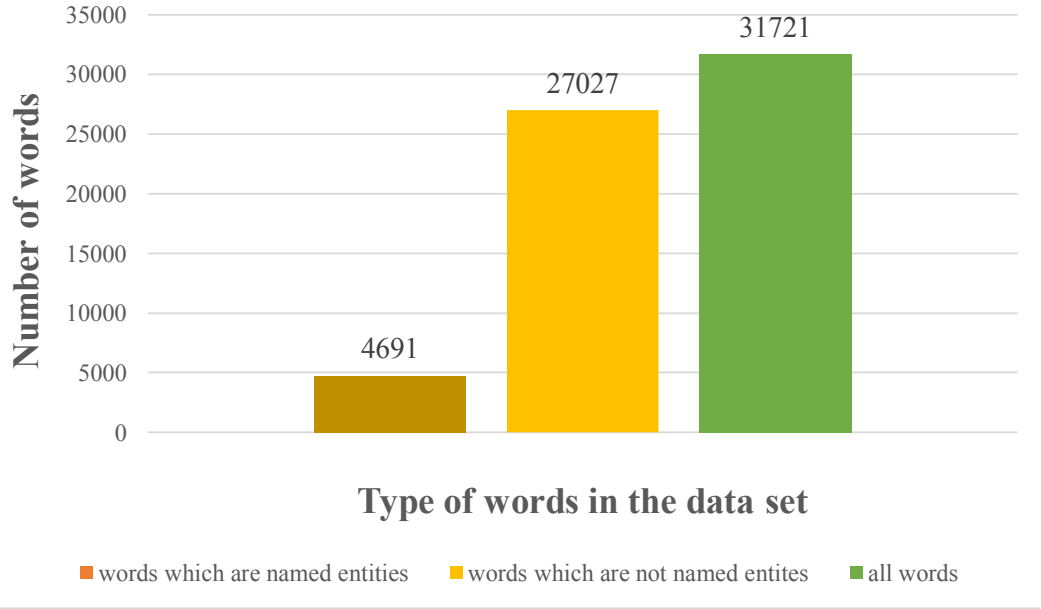


Figure 2.2: Comparison of the distribution of the types of words in the data set.

The four classes or tags that DNB is interested in are :

1. ***Company name***: The name of the company selling shares.
2. ***Meeting address***: Address of the venue of the meeting
3. ***Meeting date***: The date of the meeting in day (numeric) month (words) year (numeric) format.

4. ***Company deadline***: The last date to send the notice of attendance back to the company. The notice of attendance is a declaration stating that the shareholder will attend the meeting on the date mentioned in the letter. In case the shareholder is unable to participate in the meeting, the proxy must be sent back to the company within the deadline.

Figure 2.3 shows few rows from the data set created. In the *tag* column, the numeric part of the *tag* corresponds to the following class labels, as listed below. These numeric labels were automatically generated by the tool, discussed in Sec. 2.3.3 which was used for annotating the dataset, the numeric part and the name of the tag are used together throughout the thesis.

a) *25- company name*
b) *26- meeting date*
c) *27- meeting address*
d) *28- company deadline*

	token	file_names	id	tag	Sentence #
0	An	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	O	Sentence1
1	Extraordinary	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	O	Sentence1
2	General	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	O	Sentence1
3	Meeting	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	O	Sentence1
4	of	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	O	Sentence1
5	Techstep	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	B-25	Sentence1
6	ASA	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	I-25	Sentence1
7	will	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	O	Sentence1
8	be	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	O	Sentence1
9	held	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	O	Sentence1
10	on	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	O	Sentence1
11	4	161013 - Techstep ASA - CA 214316 - 04.11.2016...	5763	B-26	Sentence1

Figure 2.3: Few rows from the dataset

The data set consists of five columns and *31721* rows. The column *token* holds the words in the documents with one word per row, *file_names* contains information about the document to which the word belongs, *id* contains a unique identifier associated with each sentence, *tag* represents the class label of the word labelled manually following the IOB2 tagging scheme [12], *Sentence #* corresponds to the sentence number, each sentence in all the documents in the data set is labelled as a series of sentences. For example, if the whole data set contains two documents, with *document 1* containing three sentences and *document 2* containing two

sentences. The sentence numbering will be as follows: all words in *document 1 sentence1* would be labelled "*Sentence1*", all words in *document 1 sentence2* will be labelled "*Sentence2*", all words in *sentence1* of *document 2* will be labelled "*Sentence4*" and so on.

2.2 Tagging scheme

The main aim of a sequence labelling task is to extract words and to correctly label them into classes or tags. IOB is a standard tagging format used to perform chunking tasks in computational linguistics introduced by Ramshaw and Markus in 1995 [27]. Chunking is the process of dividing a sentence into an ordered set of chunks, where each chunk corresponds to word groups. Ambiguity regarding the start and end of a chunk led to the rise of the tagging scheme. A chunk can also be referred to as an entity. An entity or a chunk could be a word or group of words that belong to a class; for example, in the sentence "*Alice and John Doe love to program in python*", the word "*Alice*" is a named entity composed of a single word while the word "*John Doe*" also constitutes a named entity but is composed of two words. Both these entities belong to the class "name of a person". If the sentence is processed by a named entity recogniser interested in the classes "*name of a person*" denoted by "*PER*" and "*programming language*" denoted by "*PROG*" the word "*Alice*" belonging to the class "name of a person" gets classified as "*PER*", while the words "*John*" and "*Doe*" representing a single entity composed of two words also belonging to the class "*name of a person*" also gets classified as "*PER*" but with separate "*PER*" label corresponding to each word "*John*" and "*Doe*". The term "*python*" belongs to the class "*programming language*" and is a single word making up an entity; hence gets classified as "*PROG*".

The IOB tagging scheme makes it possible to mark boundaries of a named entity. The *I*-prefix with a tag means the word is inside the named entity, a word with an *O* tag means that the word does not belong to the named entity, *B* represents the word is at the beginning of an entity. There exist two IOB tagging schemes, namely, IOB1 [27] and IOB2 [25, 26]. In the IOB2 tagging scheme, all entities begin with "*B-tag",* be it a chunk of a single word or multiple words. A named entity composed of multiple words like *"John Doe"* starts with a *"B-tag"* followed by one or more *"I-tag"* depending on the remaining number of words in that single chunk, while a named entity composed of a single word like *"Alice"* is assigned only the "*B-tag*" (see Table 2.1). In the IOB1 tagging scheme, as shown in Table 2.2, "*B-tag*" is only

assigned to the first word in a named entity composed of a single word or more than one word if that named entity is not the first instance belonging to that class [26] in the sentence. If the entity is the first instance belonging to a class, independent of the number of words in it, it is assigned an "*I-tag*" instead. The first word will be assigned with the *"B-tag"* followed by the *"I-tag"* for the next instance belonging to the same class. Similar to IOB2 rest of the words that are not named entities are assigned *"O-tag"*. In Table 2.2, "*Alice*" is assigned "*I-tag"* as it is the first instance belonging to the class "*PER*", while "*John*" is the second instance belonging to that class. IOB2 tagging scheme is used in this study, as the python package, doccano transformer [28] used for processing the manually annotated dataset converts it to CoNLL 2003 format, which follows the IOB2 tagging scheme (see Sec. 3.2).

Table 2.1: Example of the IOB2 tagging scheme where PER- name of a person, PROG - programming language.

TOKEN	*Alice*	*and*	*John*	*Doe*	*love*	*to*	*program*	*in*	*python*
TAG	B-PER	O	B-PER	I-PER	O	O	O	O	B-PROG

Table 2.2: Example of IOB1 tagging scheme where PER- name of a person, PROG - programming language.

TOKEN	*Alice*	*and*	*John*	*Doe*	*love*	*to*	*program*	*in*	*python*
TAG	I-PER	O	B-PER	I-PER	O	O	O	O	I-PROG

2.3 Process of creating data set from PDF documents

2.3.1 Portable document format (PDF) documents

PDF, which stands for *Portable document format,* is a universally compatible file format based on PostScript format developed by Adobe Systems. It makes it possible to view documents in an electronic format independent of the system's specification. This specification includes software, operating system and hardware. Initially, it was developed as a system to share documents without losing the structure and layout, and this makes editing and information extraction from them difficult. The PDF format allows multiple types of data like text, images, videos, hyperlinks, etc making them flexible. They can also enforce different security levels for accessing the document, such as prohibiting editing or printing making them a secure file

format. The flexibility factor makes PDF files diverse, thus complicating working with them [29].

There exist three types of PDFs, namely, "True" or digitally created PDF, "Image-only" or scanned PDF and searchable PDF. Digitally created PDFs contains text and images, which gets created within applications like Microsoft Word, Microsoft Excel etc. or using the print function which routes the print demand to a virtual printer, consequently generating a PDF file. PDF files thus generated are editable in terms of changing the size, moving and deleting. There exist text that is searchable, and that is directly created using the applications mentioned above. Scanned PDF has content locked as an image. These are created by scanning a hard copy using a scanner or directly taking a picture of a document using a camera. The content could be text but is not searchable and cannot be copied as it has the text layer missing. On the other hand, searchable PDFs have OCR software (see Sec. 2.3.2) working in a scanned PDF background. Thus making it possible to search and copy the text and characters in the PDF [29].

2.3.2 Optical Character Recognition (OCR)

OCR is a text recognition technology that converts documents with text in searchable and non-searchable formats to machine readable forms, making it possible for the computer to perform searching and editing. The relevance of *OCR* arises due to the fact that being digitally accessible does not mean the data is machine-readable. This data conversion process involves identifying and converting different forms of characters like handwritten, typed, and printed text to digital format. The whole process can be divided into two major steps: text extraction from an image followed by text recognition. We can further break this down into a series of steps starting with dividing a page into blocks of texts or images followed by dividing a line in the text into words and then to characters and then finally, identifying the characters using a pattern matching algorithm. We can access *OCR* in many ways, like an *OCR* software that could be installed in a mobile device or a computer system, as a service in the cloud, as a system integrated into a scanning device, as a command line *OCR* engine, using a python wrapper for *OCR*. To improve the performance of the *OCR* system, we pre-process the scanned documents. De-skew is a typical operation performed as part of pre-processing to correct the document's alignment so that the lines in the document are perfectly horizontal and vertical.

In this study, we used a script for performing *OCR* with the help of a python package named Python-tesseract version 0.3.6 [30]. Python-tesseract is a wrapper for the Google Tesseract *OCR* engine. Google Tesseract *OCR* is an open-source *OCR* engine developed by Hewlett-Packard in 1980 and sponsored by Google since 2006. It can recognise more than 100 languages. It uses an LSTM in the background for line and character recognition [31]. The workflow of the script is as follows. First, the folder containing the documents as PDFs are fed as input to the python script, these documents are then transformed to images for each page, then de-skew is used to remove any skewing in the image, followed by image processing to identify the blocks in the text and finally run each block of text through a tesseract to get the text. The output is stored in javascript object notation line format (jsonl), where each line contains a json object. Figure 2.4 represents a single json object from the jsonl file generated as the output of *OCR*. The json object represents a single block of text which consists of three key-value pairs. Key *"text"* represents the text in a single block, *"meta"* contains the metadata related to the block of text, which holds information about the *x* and *y* coordinate of the block based on its alignment of the block in the page, the height and width of the block as well as the page number corresponding to the page number in the file to which the text belongs. The filename from which the block of text is extracted is included in the key *"file_name"*. Page segmentation mode (*"psm"*) = 6 denotes that we consider each block as a single block of text during the processing. There exist multiple configurations for page segmentation mode supported by tesseract based on the value assigned to psm [32].

{"text": "The shareholders of X ASA are hereby given notice of the extraordinary general meeting to be held on 29 November 2016 at 09.30 Oslo time, at Munkedamsveien 35 (15 floor) in Oslo. ", "meta": {"x": 362, "y": 520, "w": 1825, "h": 114, "psm": 6, "lang_used": "nor", "page_no": 1, "file_name": "X - CA 216386 - 29.11.2016.pdf"}}

Figure 2.4:Example of a single json object from the jsonl file created by OCR

2.3.3 Annotation tool

Doccano [33] is a text annotation tool that is available for free. It aids in manually creating labelled data set for various natural language processing tasks like text classification, sequence labelling and sequence to sequence learning. Text classification involves the process of

classifying text into predefined classes based on the content of the text e.g. in sentiment analysis the text could be classified as a positive, negative or a neutral text based on the sentiment. Another example would be topic labelling where the text can be classified based on a major topic like "Science" focus such as, if the text is about a scientific theory. Similarly, for spam detection where the text could be classified as spam or not spam depending on certain criteria about the content of the text. Sequence labelling task deals with assigning a label to each element in a sequence like named entity recognition while sequence to sequence task is a process of converting a sequence of words in one domain or language to a set of sequence in another domain or language, e.g. language translations. [33]. The jsonl file obtained from the *OCR* is fed as input to *doccano*. The output from *doccano* for annotation created for a sequence labelling task in json format is shown in Figure 2.5. The json output contains information about the annotations created in the text. The *"annotation"* key holds information on the label assigned to the word and with the start offset, which denotes the index at which a labelled word starts and end offset denoting the index at which the labelled word ends.


```
{"id": 5841, "text": "INFORMATION CONCERNING SOLICITATION AND VOTING FOR THE ANNUAL GENERAL M
EETING OF SHAREHOLDERS (THE "MEETING") OF 2020 BULKERS LTD TO BE HELD ON NOVEMBER 13, 2018",
"annotations": [{"label": 25, "start_offset": 113, "end_offset": 129, "user": 1, "created_a
t": "2021-01-13T02:55:11.567313Z", "updated_at": "2021-01-13T02:55:11.567374Z"}, {"label": 2
6, "start_offset": 144, "end_offset": 161, "user": 1, "created_at": "2021-01-13T02:55:16.2400
45Z", "updated_at": "2021-01-13T02:55:16.240120Z"}], "meta": {"x": 588, "y": 259, "w": 1298,
"h": 213, "psm": 6, "lang_used": "nor", "page_no": 2, "file_name": "2020 Bulkers Ltd-Notice a
nd Proxy - CA 272205 - 13.11.2018.pdf"}, "annotation_approver": null}
```


Figure 2.5: Example of annotated output from doccano in jsonl format

2.3.4 Processing the annotated data

Doccano transformer, an open source tool available from GitHub [28], helps in converting the data in json format into a format based on our requirement. It has two format options CoNLL 2003 and spacy. We are using the CoNLL 2003 format in this study. The CoNLL 2003 format converts the data into IOB2 format. The output of the *doccano transformer* contains a tag associated with each word. This output is in the form of a generator object which can be converted to a list of dictionaries. These dictionaries contain *"data" as a key* that holds the text in each line along with the tags in IOB2 format. The value corresponding to *"data"* is processed to remove the unwanted spaces and symbols in it and convert it to a data frame with four columns, namely token, tag, sentence number, filename etc.

2.3.5 Process flow

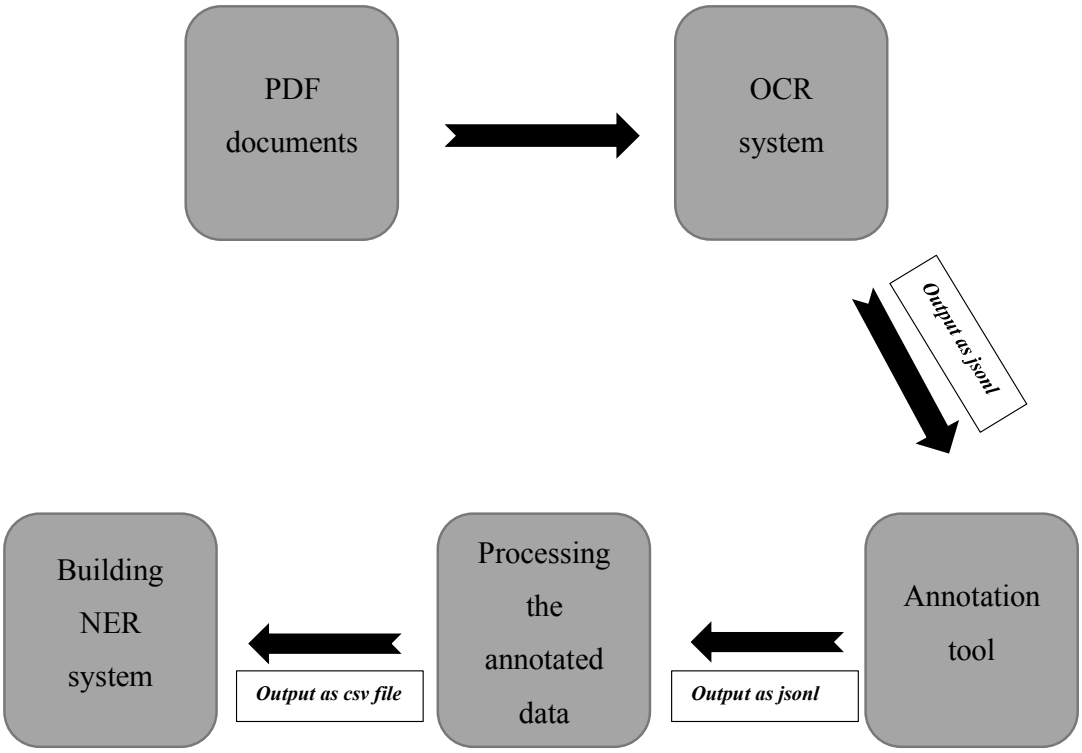


Figure 2.6: Process flow chart showing the steps involved in converting PDF documents to a format suitable for training, where jsonl - java script object notation lines format, csv - comma separated file format, PDF - portable document format, OCR - Optical Character Recognition

Figure 2.6 shows the process flow followed in extracting text from PDF documents and creating data sets to build and evaluate an automatic NER model. The PDF documents were processed using *Google Tesseract OCR,* which aided the process of extracting text from the PDFs. The output of *OCR* in jsonl file format, as shown in Figure 2.4, was fed as input to an open-source text annotation tool, *doccano* [33], for manually annotating the text thus, enabling the creation of the gold standard data set for the named entity recognition task. Gold standard data set refers to the data set containing the ground truth word-label mappings; here, the gold data set is one that is manually created. The annotation tool was set up using a docker container as per the git hub page's instructions. A new project was created after logging into *doccano* for sequence labelling tasks. The labels of our choice were defined as shown in Figure 2.7, the output of *OCR* as jsonl file was imported, and the process of annotating the data was initiated, as shown in Figure 2.8. At the end of the annotation, the output was exported as a javascript object notation (JSON) file. This output from *doccano* contains an additional field called "*annotation*" that includes the label and other information about the text that is labelled like the start index and end index of the entity and the information from the *OCR* output. This output

was further processed to convert it to a format suitable for feeding as input to the NER model. This stage takes into account the tagging scheme (see Sec. 2.2). The processed data was then used to build an automatic NER system using the deep learning models as described in Chapter 3. The software versions and the code used for *OCR* and transforming the *doccano* output to a CSV file suitable for feeding as input to models is encapsulated in tables (Table 2.3 and Table 2.4). The repository also contains the folder containing the documents that were processed by *OCR*, output of *OCR*, output of *doccano* and the final data set.

Table 2.3: Filename and githash of the file containing the code to transform jsonl downloaded from doccano to a format suitable to be used as NER dataset available from https://github.com/meerajsph/Master_thesis, this also includes the ocr code used and the prerequisites to run the ocr code

File	Githash
convertjsonfromdoccano_to_trainingdata.ipynb	515e5ab
ocr_and_doccano	c118c72

Table 2.4: Name and version of software used for transforming the output of doccano to a format suitable to be used as input data for NER

Software	Version
Doccano transformer	1.0.1
python	3.6

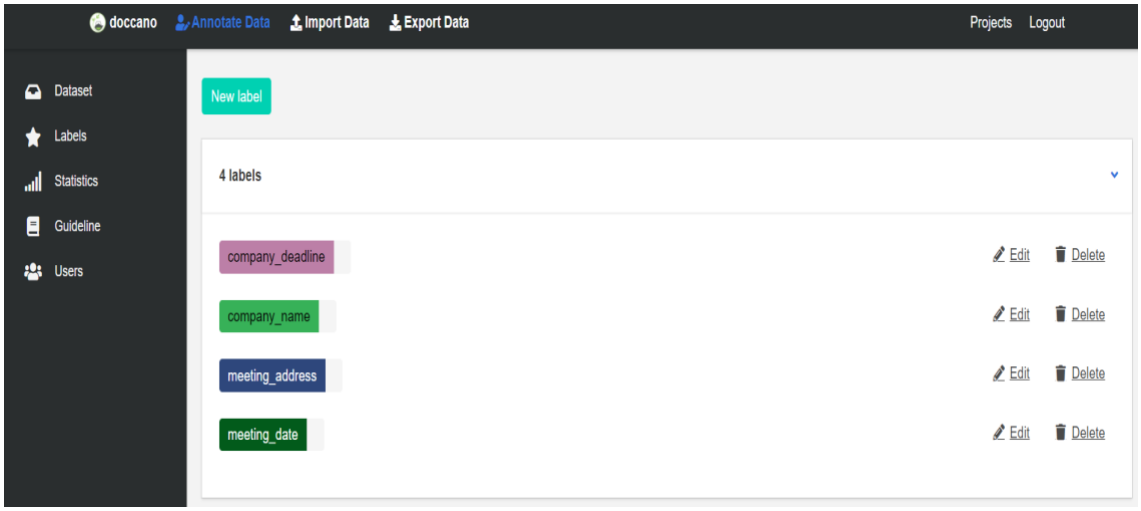


Figure 2.7:Creating labels in doccano

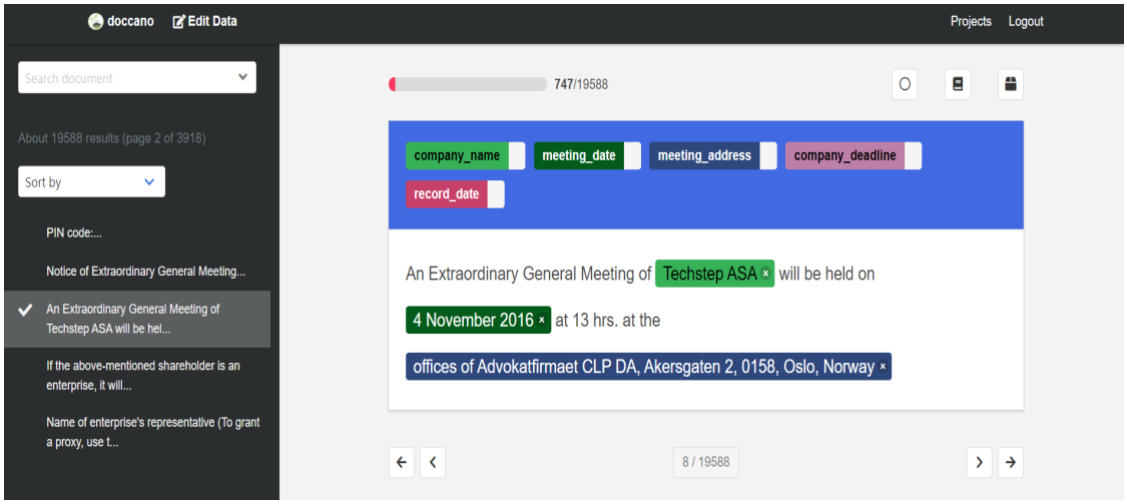


Figure 2.8:Example of the process of text annotation in doccano

Chapter 3 Methods

In this section, we describe the methodology used for solving the problem of automatic information extraction from unstructured data in the form of pdf documents at DNB. The methodology begins with an introduction to the solution to the problem, which is by Named entity recognition (NER) [34]. The section is further divided into data pre-processing, building the model, training, evaluating, and comparing the performance of automatic NER systems built using deep learning models based on Bi-directional Long Short Term Memory Conditional Random Field (Bi-LSTM-CRF) architecture when coupled with different word vectorisation techniques [35]. Additionally, this includes the process of optimising the hyperparameters of the best performing model found by the comparative study to build an efficient NER system that works well with DNB data. The theory behind the various components used in the model architecture, including the word vectorisation techniques, optimisers, activation function, loss used to train the model and the different layers involved, are also discussed.

3.1 Named Entity Recognition(NER)

The problem of information extraction from unstructured data can be solved by using Named Entity Recognition (NER) [34]. The term “named entity” was first coined in 1996 at the 6th Message Understanding Conference (MUC6) [34] organised by Navel Research and Development group (NRaD), RDT&E division of Naval Ocean systems Centre (NOSC). The NOSC started MUC to assess and help develop research in the field of automated analysis of textual data in military messages. These conferences are not like the traditional ones but are organised to help promote research and development in the field of information extraction by making the participants carry out practical experiments before attending them. The participants were given sample messages and were directed to build a system for information extraction. The instructions regarding the type of information to be extracted was also shared with them beforehand. The efficiency of the systems built by them was evaluated by running a set of test messages and comparing the output with the answer key corresponding to those messages. These answer keys serve as the golden standard or expected result of information extraction. While defining the main goals for the conference named entity recognition was introduced as

a “short-term subtask”. The main goal was to find practical, domain-independent information extraction systems from those developed by the participants, which succeeds to perform with high accuracy. These technologies were evaluated based on their ability to identity name, organisation and location from a text. This identification subtask was referred to as “named entity recognition” by the committee [34]. However, the first reference of NER was earlier in 1991 when Lisa F. Rau described an algorithm for extracting company names from financial news. It was suggested that identifying these names could help solve the problem of the presence of unknown words and can be used for topic analysis, information extraction, database generation and database querying etc [36].

A NER task aims to extract important words or expression from a given text. There exist several definitions for a named entity in various research papers. These definitions are ambiguous and unclear. In a paper by Marrero et al. [2], the authors highlight how the term named entity is described in different researches. It highlights the fact that there exists a disagreement about what a named entity is. Analysing these definitions allows them to categorise a named entity based on four criteria: if the word is a proper noun, a unique identifier, has a rigid designator, and is based on purpose or domain.

Proper noun refers to a name specific to a person, place or thing, e.g. New York. This concept was put forward by Petasis and colleagues in 2002 [37], but it was found to be insufficient to define an named entity due to absence of inflexions and lexical meaning in addition to the fact that a proper noun should begin with capital letter. Kripke [38] introduced the concept of rigid designator in the theory of names. It refers to the philosophy that the same word is used for an object in all context in which it exists, e.g. he suggested that the name “Richard Nixon” is a rigid designator for the U.S president. But this concept has been quite controversial because the president of U.S.A keeps on changing and Richard Nixon would not refer to the president at all times. A unique identifier is a unique label that refers to the entity, e.g. the term “1 million $” for this to be a named entity this has to be a unique identifier independent of the context in which it is placed. But it was found that it varies with context as it could be American dollars or Mexican pesos depending on the context. Purpose or domain of application refers to area or field of study to which the entity belongs, e.g. the word “water” belongs to the domain “Chemistry”. In contrast “Batman” could belong to the domain “movies” as well as “costumes”. This criterion, based on the four factors, was used to analyse the annotated corpora or guidelines presented by various NER conferences or publications such as the Seventh

Message understanding Conference (MUC-7) [39], Sekine's Hierarchy [40], Computational Natural Language Learning (CoNLL) [20], Automatic Content Extraction (ACE) [41], GENIA [42] etc. But it was found that the entities listed by these conferences or publications did not satisfy the proposed four categories. This "ambiguity" in the definition of NER also has consequences in the way different NER tools function. There exist differences in the type of information that is considered as a named entity in different tools. Although there are common categories like the name of a person, organisation, and location, several other entities are exclusive to the tool [2].

NER is a subfield of Natural Language Processing (NLP), serving as a down stream task for various NLP applications [43], such as Information Retrieval [44], Machine Translation [45], Summarisation [46] or Question Answering [47]. Natural language processing refers the process of giving the computer the ability to understand natural or human language while NER can be defined as a process of extracting important information from textual data sources and classify the information into predefined classes.

There exist studies which suggest that the problem of NER is solved based on the precision and recall scores obtained for certain experiments for performing information retrieval. In a study by Marrero et al. [2], the experimental validity was analysed. Experimental validity refers to how well an experiment fulfils the requirement. This evaluation was done based on four types of experimental validity namely content validity, external validity, convergent validity and conclusion validity and it was found that the experiments did not meet the validity criteria. Additionally, each forum's tools and evaluation metrics for evaluating the results of the experiments are distinct, thus comparing the metrics generated by them cannot be considered a valid comparison. NER problem has been considered to be solved based on their performance in selected domains like news. But these results are not transferable to other fields which trigger the need for the development of NER in other domains. The unavailability of annotated data sets for training the NER model could be considered as a major reason for this drawback. Therefore, it cannot be said that NER is a solved task [2, 48, 49].

3.2 Pre-processing the data

This section describes the steps involved in preparing the dataset collected as described in Sec. 2.1 and transforming them to forms suitable for feeding to a deep learning model for performing NER. The actual data preparation step includes the process of converting the words to vectors known as word vectorisation [35] (see Sec. 3.3.6). But, the word vectorisation step occurs during the process of building the model as discussed in Sec. 3.3. Thus, this section includes the pre-processing steps involved to make the data suitable for the word vectorisation technique, which is the first step in building a deep learning model for NER.

3.2.1 Splitting the data set into training data, for building the model and test data, for validating the model

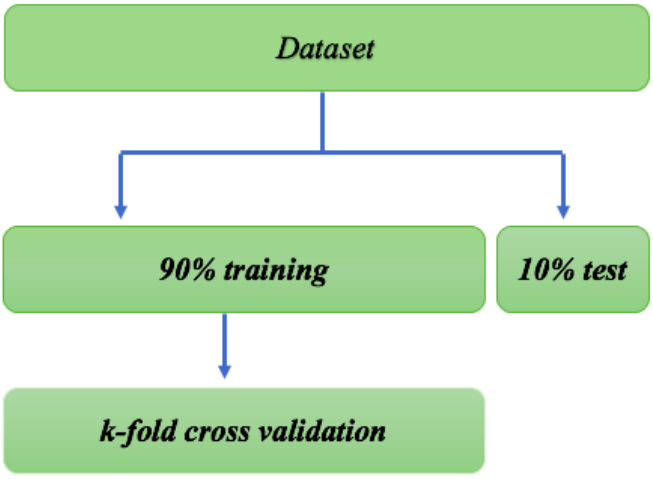


Figure 3.1:Process of splitting the data set

Data in comma-separated values (CSV) file format with three primary columns: *token*, *tag*, and *sentence number*, as explained in Sec. 2.1 was used as the input. This data was then grouped based on the sentence number, consequently transforming each input into a list of words belonging to a sentence, forming a list of individual lists corresponding to each sentence as well as a list of lists consisting of tags or labels corresponding to the words. The words and tags are grouped sentence-wise to preserve the context of the words while feeding them as input to the deep learning model. This data was then divided into features and target represented by X and y, where X contains the list of lists of words and y contains list of lists of tags. We convert all words in X to lowercase to make the word representation precise, else a word in uppercase letter and lowercase would be considered as two different words and would be

handled differently. The data set was then divided into three parts as training data, validation data and testing data (see Figure 3.1). The purpose of dividing the data into three parts was to avoid information leak while validating the model, and hence Scikit-learn train test split was used for the purpose. The whole pre-processed data set was split into *90* percent training data and *10* percent test data. The randomness of the split was fixed by setting the random state to a value of *42*. The training data used for training the model was further split into training and validation data by *k*-fold cross validation, as explained in Sec. 3.4.4. The model was evaluated on the validation data to find the number of epochs at which the model gave the best performance. The whole training data was then fitted on the best model, training it for the *e* number of epochs, where *e* represents the epoch number at which the model gave the best results. Finally, predictions were made on the test data, and the F1-score was calculated on the test data. Additionally, while using ELMo embeddings, the model required data sets in sizes divisible by the batch size. Hence we sliced the data set to be in this size range.

3.2.2 Tokenising and padding

Neural networks need input in the form of *multi-dimensional* vectors composed of numbers, since they cannot work directly with words. This was hence achieved by a process known as word vectorisation, which is explained in (see Sec. 3.3.6). The majority of word vectorisation techniques used in this study requires words to be converted to unique integers before transforming them to vectors except for word vectorisation using Embeddings from Language Models (ELMo) [13]. The experiments were performed in Google colab [50] using TensorFlow and Keras API. TensorFlow is an open-source software and is one of the most popular libraries used for building deep learning models [51]. While Keras is a high-level API which is developed to interact more easily with TensorFlow. The packages used and their versions are reported in Sec. 3.4 along with the GitHub link to the full code used.

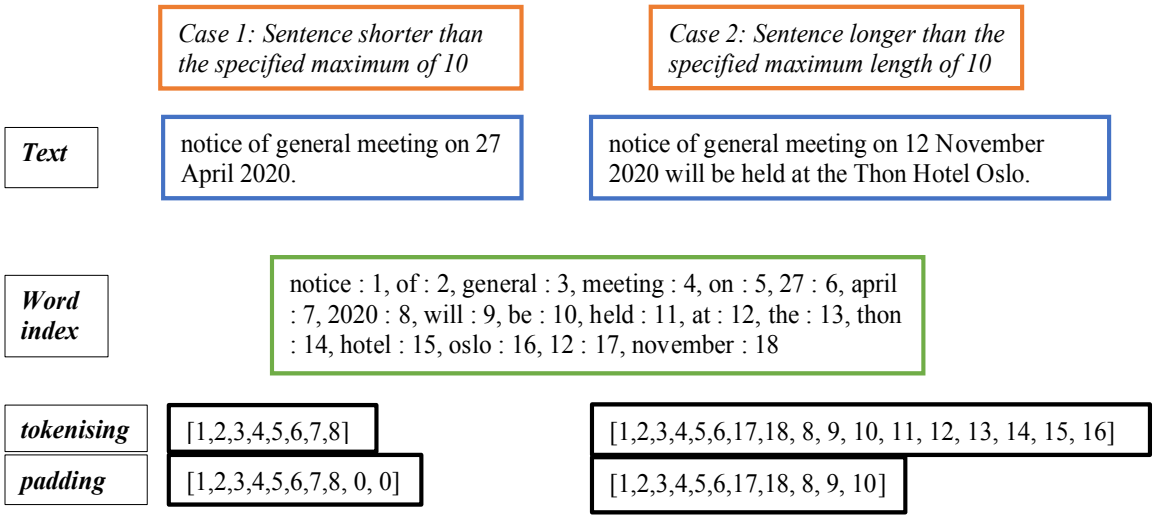


Figure 3.2: Example of the process of tokenising and padding sentences for a specified maximum length of 10, word index represents the dictionary created by Keras tokeniser creating an index corresponding to each unique word in the data set

Tokenisation is performed by TensorFlow Keras which comes with the functionality to convert unique words to integers by using TensorFlow Keras tokeniser [51]. This tokeniser was trained on the training data to avoid information leakage between the training and the test data. The tokeniser object created a dictionary representing each unique word as a key and the index as their value. The words which occur most frequently were assigned the lower indices. The index number starts from one to the total number of unique words in the training data. An index value of one is reserved for a word that is outside the vocabulary of training data. This tokeniser object trained on the training data is then used for converting the test data to integers. If the test data contains words that are outside the training data's vocabulary, the words are replaced by the *"OOV"* token which stands for Out of Vocabulary and are assigned an index of one, which is reserved for "*OOV*" by Keras tokeniser. This enables us to maintain the sentence length without skipping the words outside the training data set's vocabulary.

Tokenisation is followed by padding. Each sample, represented by a list of integers after tokenisation, must also be of the same length before feeding it to the deep learning model. Hence all sentences were made to have a specified maximum length. This specified maximum length was decided based on the length of the majority of the sentences in the corpus. A specified maximum length of *100* words was selected for the study based on the sentence length distribution in the data set as shown in Figure 3.3 as most sentences have length below *100* words, a value of *100* was also chosen to reduce the loss of words or entities from the data

set. Padding is a process of making all sentences to the same length by adding a pad word or index corresponding to the pad word at the end or beginning of the shorter sentences to make it to the specified length [52]. During padding, the pad word is assigned an index of zero by Keras. The location of pad word can be specified in the function call. For sentences longer than the specified length it was cut to this specified maximum length. Both these can be achieved by using Keras built-in method known as “pad sequences”. An example of both these steps during two possible situations are depicted in Figure 3.2 for a better understanding.

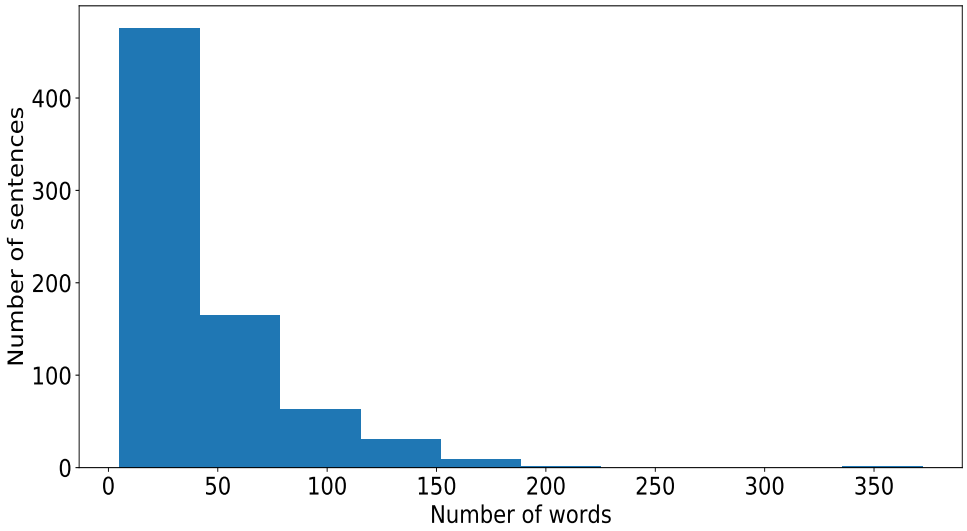


Figure 3.3: Distribution of sentence length in the corpus, where the count plot illustrates the number of words versus the number of sentences.

The tag words which are categorical variables that classify each word to a label or tag from the unique set of tags. These tags are also tokenised and padded, the tags are tokenised by mapping each unique tag to a dictionary of integers starting from one to the number of unique tags in the training data. Tags are padded with "*O*", which stands for a word that is outside the named entity and are assigned the tag index corresponding to "*O*" tag (Sec. 2.2). The tags associated with *“OOV”* words are also assigned the "*O*" tag and the corresponding tag index. The integer corresponding to the tags are converted to a binary matrix similar to one-hot encoding with the help of Keras in-built categorical function. The process involved is known as categorical encoding. There exist two types of encoding of categorical variables namely, label-encoding and one-hot encoding. Label encoding involves the process of converting categorical variables to integers based on alphabetic order of the tags. One-hot encoding on the other hand, creates features corresponding to a categorical variable based on the unique number of categorical

variables. One-hot encoding results in a vector composed of only zeroes and ones corresponding to a tag, where one denotes that the word corresponds to the tag at that location where one is present and rest of the places are assigned zero. The dimension of the vector depends of largest value in the tag dictionary mapping tags to integers. Keras categorical function creates a binary matrix of dimension equal to the largest index value in the *tag dictionary +1*. For example, if there are four tags and a word in the sentence belongs to the second tag, the tag index dictionary will be of the form dictionary [*"class1" :1, "class2" :2, "class3" :3, "class4" :4*] then the Keras categorical function maps tag "*class2*" to [*0, 0, 1, 0, 0*]. One-hot encoding is preferred to use when the categorical features do not have an order and when the number of categories are not so large [53].

3.3 Building the model

This section discusses the model architecture. The model architecture can be broken down into three modules: the input layer, which takes the sequence of words as input, the embedding layer, composed of the word vectorisation techniques, the hidden layer which includes the deep learning model followed by an output layer, which in this study is a conditional random field layer. To understand the architecture used for building the NER models, each model component and techniques used during the process are described in the section below.

Theory behind the model components

3.3.1 Activations functions

Deep learning is a subfield of machine learning where neural networks are used to build the deep learning model [54]. The term deep neural network refers to a network composed of many layers where each layer is composed of multiple nodes known as neurons (see Figure 3.4)[55]. As the number of layer increase, the deeper the network. These nodes have an activation function also referred to as transfer functions which are used to decide if a neuron gets activated and how much it gets activated based on the net input. The net input is the weighted sum of input and bias fed to a node given by the *Eq. (3.1)*.

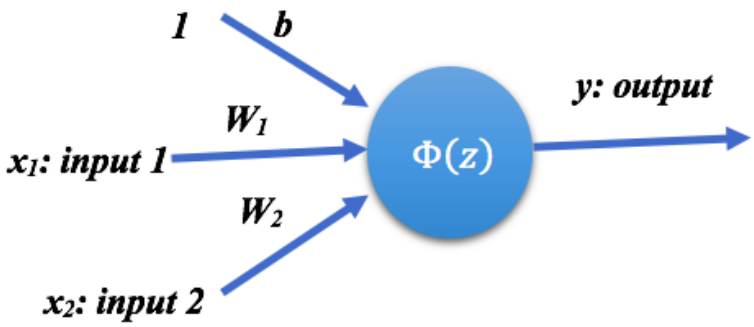


Figure 3.4: Single neuron representation with two inputs, where x_1, x_2 are input values and w_1, w_2 are weight vectors associated with it, b is the bias , y is the output value, $\Phi(z)$ is the activation function, z is the net input

$$z = \boldsymbol{W_1 x_1 + W_2 x_2 + b} \qquad (3.1)$$

Where:

z : *Net input*

$\boldsymbol{x_1, x_2}$: *input values*

$\boldsymbol{W_1, W_2}$: *weight vectors associated with each input*

If an activation functions is not present in a neural network each neuron will only perform linear transformation on input and will make the network simple but incapable of learning non-linear features from the data. There exist linear and non-linear activation functions but selecting an activation function depends on the problem we are trying to solve using a neural network as well as type of layer and the expected output from the particular layer [56].

Sigmoid activation function

The sigmoid activation function, also known as logistic activation function is a non-linear activation function that is employed in binary classification problems. Based on the value of input fed to the node with the activation function it shrinks the net input to values between *0* and *1* (see Figure 3.5). This value represents the probability of belonging to the first class in a two class classification task [56]. It transforms input values less than *0* to values between *0* and *0.5* and those greater than *0* to values between *0.5* and *1*, where a net input of *0* gets transformed to a value of *0.5*. The sigmoid activation is given by the equation *Eq. (3.2)*.

$$\Phi(z) = \frac{1}{1+e^{-z}} \tag{3.2}$$

$\Phi(z)$: *represents the sigmoid activation function*

z : *represents the net input*

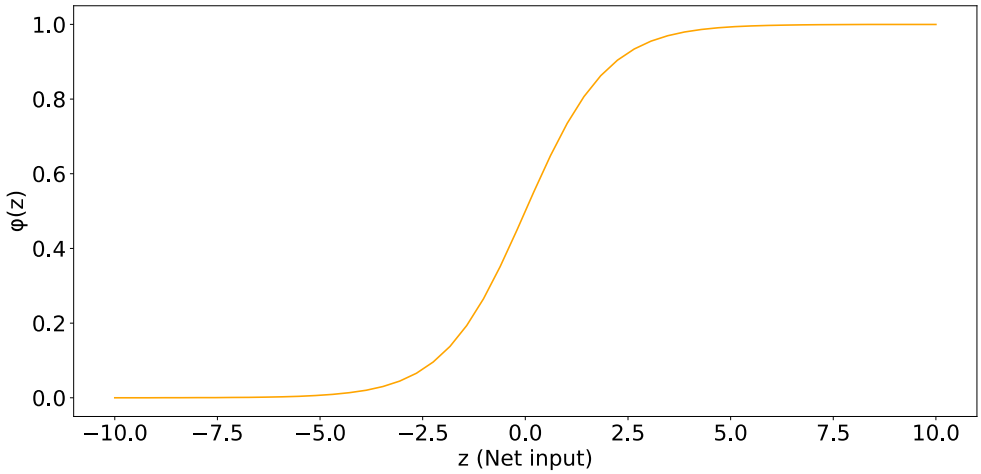


Figure 3.5: Sigmoid activation function

Hyperbolic tangent (tanh) activation function

A hyperbolic tangent (see Figure 3.6) is also an s-shaped activation function similar to a sigmoid used in deep learning that transforms the input values to values between *-1* and *1*. It transforms input values less than *0* to values between *0* and *-1* and values greater than *0* to values between *0* and *+1*. Tanh function became preferred in comparison to a sigmoid activation function in deep neural networks based on their better performance in terms of number of epochs needed to minimise the loss and improve the accuracy [57, 58]. Tanh activation function is represented by *Eq. (3.3)*.

$$\Phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.3}$$

Where $\Phi(z)$: *represents the hyperbolic tangent activation function*

z : *represents the net input*

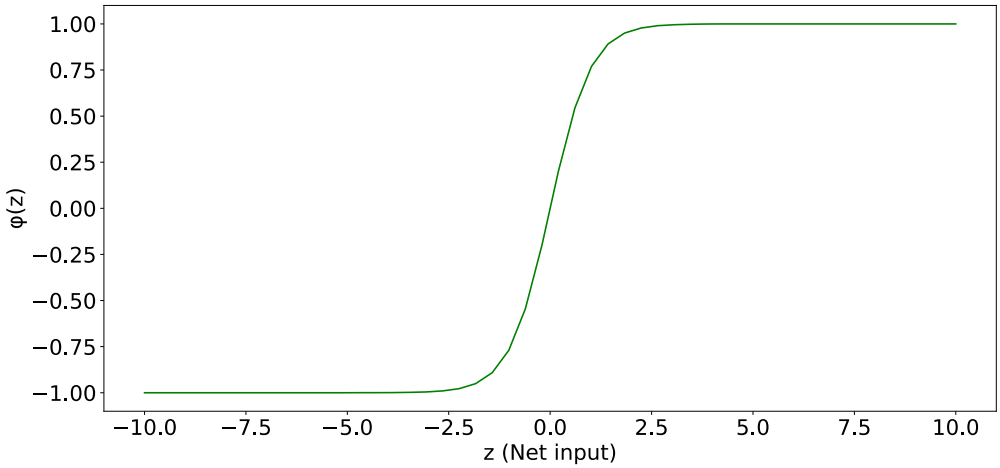


Figure 3.6: Hyperbolic tangent activation function

Softmax activation function

Softmax is a generalisation of the sigmoid activation function which produces output as a probability distribution. The primary difference between a sigmoid and a softmax activation function is that sigmoid is used for binary classification problems while a softmax is used for multi-class classification [56]. The output of a softmax consists of a *k-dimensional* vector, where *k* is the number of labels. Each word in a sentence would have a corresponding *k-dimensional* vector. This vector represents the probability of a word to belong to a label. The sum of values in the vector representation would be one. Based on the index corresponding to the location where the maximum value occurs in the vector, a tag gets predicted. The softmax activation function for a net input is computed using the equation *Eq. (3.4)*.

$$\Phi(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \qquad (3.4)$$

Where z : *net input to the softmax function*

z_i : i^{th} *element of the input vector*

$\Phi(z_i)$: *softmax activation function*

k : *number of classes or labels*

Rectified Linear Unit (ReLU) activation function

There exist several activation functions, and some, such as the sigmoid and tanh should not be used at the hidden layer because of the vanishing gradient problem [59]. In a neural network, a model is trained by forward propagation and backward propagation. During backward propagation, the derivative of the activation function at each node becomes relevant as weights are updated at a node based on the gradient of the loss with respect to the weight. The gradient of the loss with respect to the weight is calculated by the chain rule, which involved multiplication of the derivative of the activation functions. Since the derivative of sigmoid and tanh can be really small for large positive and negative inputs it can lead to a really small value of gradient leading to extremely small update of weight and thus resulting in a scenario where the weights are no longer getting updated. This makes the model incapable of learning and making correct predictions.

In this study we are using ReLU [60] as the activation function in the hidden layer. It sets the negative values in the output to zero and positive values remains the same (see *Eq. (3.5)*). It also has a derivative value of *1* for all positive values of input (see Figure 3.7) and zero otherwise. ReLU is not used in the output layer as they can lead to dead neurons in the output leading to no gradients at that particular output [61, 62] thus making the output meaningless. Though ReLU creates dead neurons in hidden layer as well, they produce higher value of gradients for positive inputs in comparison to the previous activation functions such as *sigmoid* and *tanh* thus making them less prone to vanishing gradient problems for positive values of input and models tend to converge faster with ReLU [63]. Therefore, ReLU is most commonly used in the hidden layers of a deep neural network.

$$\Phi(z) = max\ (0, z) \qquad (3.5)$$

Where $\Phi(z)$: *represents the ReLU activation function*

z : *represents net input*

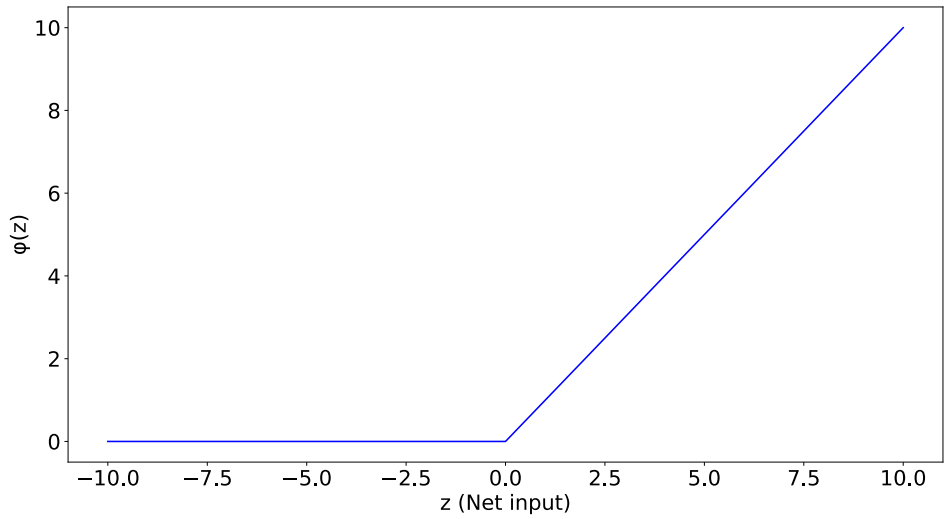


Figure 3.7: ReLU activation function

3.3.2 Recurrent Neural network (RNN)

In a NER task, we consider the data to be sequential. A sequence in mathematical terms is a collection of objects or terms where the order in which they occur is important [64]. Figure 3.8 shows the process of how sequential data is handled, where the input data representing a single sample or sentence is considered as a sequence of words like <x_1, x_2, x_3, etc.>. Each value x_1, x_2 etc. corresponds to the words occurring in the sample and the *1, 2, 3*, etc denotes the position of the word in the sentence [55]. This position of words also relate to the time step, which denotes the point in time when that particular input is fed to the model. First the first word is fed to the model at *time step = 1* followed by the second word at *time step = 2*. Since only one word is fed to the model at a time, the concept of *time step* is used to explain the process.

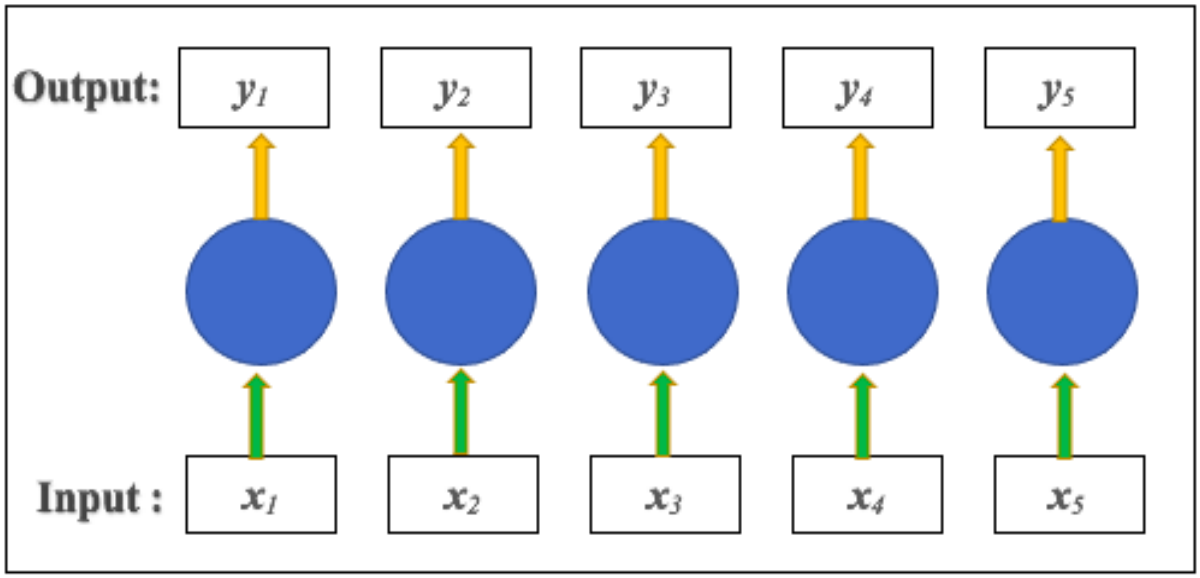


Figure 3.8: Sequential data given as input to an RNN

There exist different variations of sequence modelling tasks depending on the relationship between the input and the output as shown in figures (Figure 3.9, Figure 3.10, Figure 3.11, Figure 3.12) below. Many-to-one sequence modelling (Figure 3.12) is used in applications like text summarisation, sentiment analysis etc., where one output value is predicted by the model based on a sequence of inputs. One-to-many (Figure 3.9) sequence modelling involves sequence prediction based on a single input to produce an output sequence composed of a collection of elements, e.g. image captioning where the input to the model is a single image, and it generates a sequence of words as the caption. Many-to-many a sequence modelling task where a sequence of words is given as input to the model and a sequence with the same length as the input is generated. It could be synchronised or delayed. An example of a synchronised many-to-many model (Figure 3.10) is named entity recognition where for each word in a sentence labels are generated. In contrast, an example of a delayed many-to-many model (Figure 3.11) is language translation, where the whole sentence has to be processed by the model before translating them.

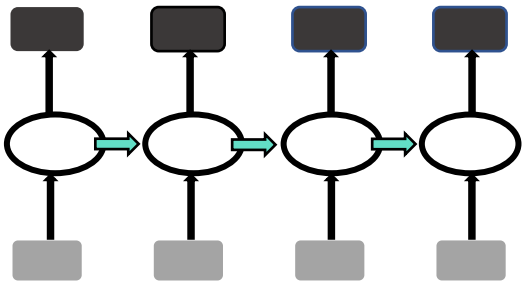


Figure 3.10: Many-to-many synchronised sequence modelling

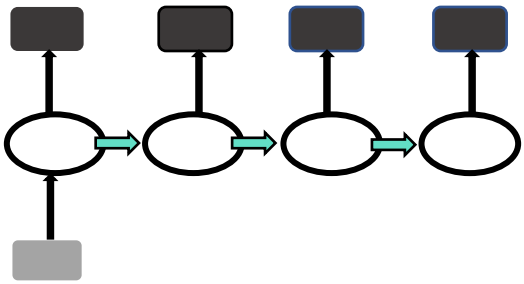


Figure 3.9: One-to-many sequence modelling

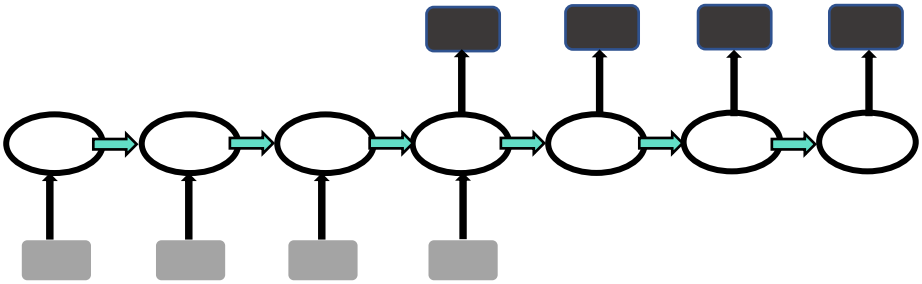


Figure 3.11: Many-to-many delayed sequence modelling

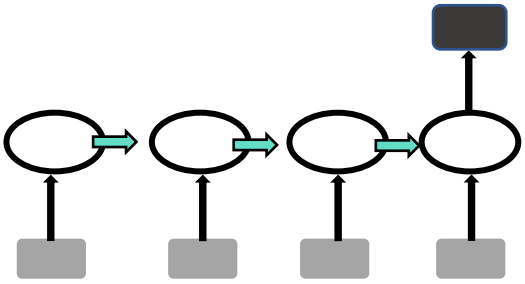


Figure 3.12: Many-to-one sequence modelling

For this study, we consider the many-to-many sequence modelling task implemented using an RNN [65] for performing NER. The reason for selecting RNN for sequential modelling is because it takes into account the order in a sequence and captures the past information, this order is denoted in terms of timesteps. The sequence is represented by a single sentence. A single word is fed as input to the model at a particular time step. The next word is fed to the model at the next timestep, after the first word. A multi-layer perceptron, on the other hand, does not consider the order in which the words occur in a sequence, and they assume that every word is independent of each other. Therefore in situations where either input data or output

data is a sequence, we consider it as a sequence modelling task. Additionally, in a sequence modelling task if a sequence of words represents the input to the model, a single word is fed to the model at a time.

The RNN architecture comprises three major layers: an input layer, a hidden layer, and an output layer (see Figure 3.14, Figure 3.15). The input layer takes as input a sequence of words. In an RNN, the hidden layer consists of a recurrent edge, showing that it gets input from the input layer at the current time step and the hidden state from the earlier time step [55], as shown in Figure 3.14. The hidden state from previous timestep is obtained by applying the activation function at the hidden layer to the input from the previous timestep. The hidden state from the previous time step thus contains information about words that occurred in the previous time steps, making it possible to capture past information. The output layer returns a sequence as the output. The same architecture is repeated for each word or each timestep in the sequence. RNN can be a single layer RNN (Figure 3.14) and multilayer RNN (Figure 3.13), depending on the number of hidden layers. The process of information flow is similar in both except for an additional step due to the presence of additional hidden layers in case of a multilayer RNN.

In a single layer RNN at *time step = 1* the weights of the hidden layer is randomly initialised. The hidden layer receives input from the current time step as well as the hidden state from the previous time step. On the other hand, in case of a multilayer RNN in addition to initialising the hidden state at the *time step = 1* with random numbers or zeroes and receiving input at the current time step as well as the hidden state from the previous time step the output from the first hidden layer is forwarded to the next hidden layer. The second hidden layer receives the output from the first hidden layer unit at that time step as well as the hidden state from the previous time step. It should be noted that during this process of information flow from the first time step to the last time step RNN suffers from the problem of information loss, due to which the hidden state received by the hidden layer at the last time step fails to contain all the information from the first time steps. This problem of information loss occurs due to a phenomenon known as the vanishing gradient problem [59] which makes it difficult for RNN to handle long-term dependencies. This is explained in detail below in the challenges faced by RNN part.

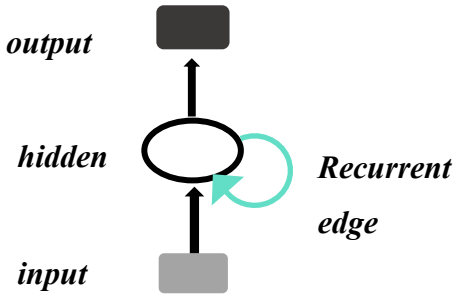


Figure 3.14:Compact architecture of single layer RNN

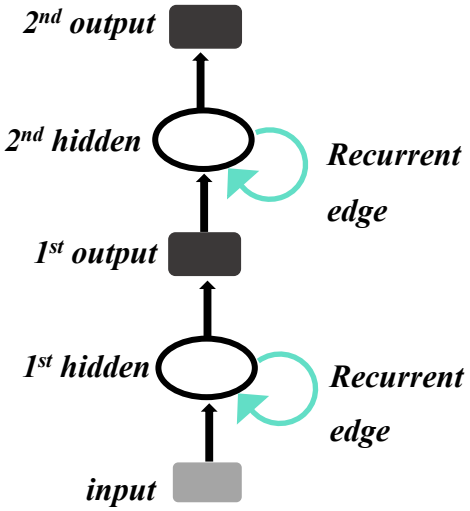


Figure 3.13: Compact architecture of multilayer RNN with the output from the first hidden layer is forwarded to the next hidden layer and the hidden layer at the next timestep

In a NER system, the input layer corresponds to the input features, which is the input sentence consisting of a sequence of words. The sentence is a collection of tokens or words, and the output of output layer is a sequence of tags corresponding to each word. Figure 3.15 shows the unfolded architecture of a single layer RNN used for NER where each word corresponds to the input at a time step *t* to the input layer. These input features which are in the form of a text can be converted to numbers by the methods described in Sec. 3.3.6. These input features are then classified as company name, meeting address, meeting date, company deadline with the corresponding tagging scheme at the end of the output layer. The output layer can be a softmax classifier (see Sec. 3.3.1) [56, 66] for predicting the probability distribution over the labels at a time step, for example, if there exist four classes. The output at each time step would be a *4-dimensional* vector with values between *0* and *1*. The sum of the values in the *4-dimensional* vector will be one. Each value as a *4-dimensional* vector corresponds to the probability to belong to a class. Suppose the value at the third place in the vector is the highest among the four. It can be inferred that the input word belongs to class three.

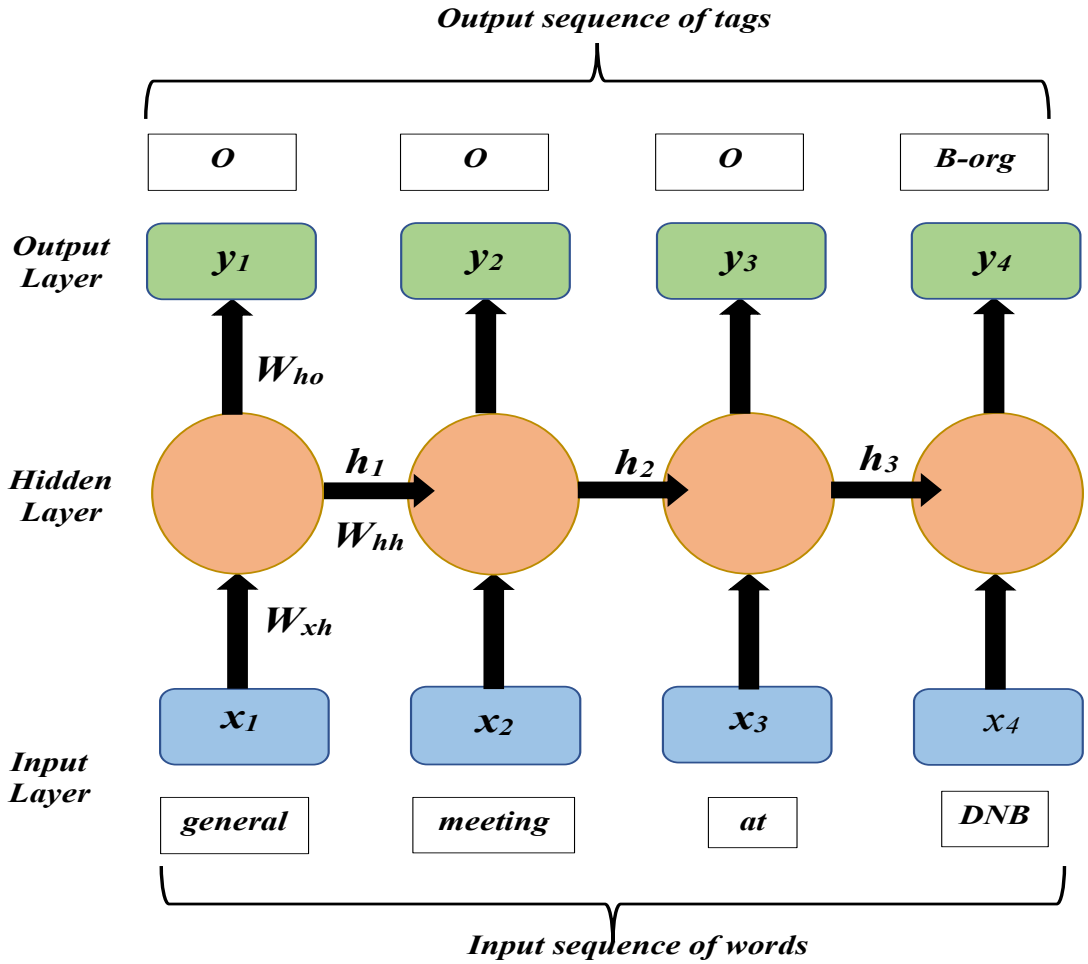


Figure 3.15: Unfolded Single-layer RNN architecture, where org: name of an organisation, B-tag, O-tag according to IOB2 tagging scheme, x_1, x_2, x_3, x_4 *are the inputs at each time step and* y_1, y_2, y_3, y_4 *are the corresponding outputs,* W_{xh}*: weigh matrix between input and hidden layer,* W_{hh}*: weight matrix between the hidden layers,* W_{ho}*: weight matrix between hidden and output layer.*

Working of RNN

In an RNN the flow of information occurs by forward propagation from input layer to output layer through the hidden layer similar to a feed forward neural network. It gets input from the input layer at the current time step. In addition, it has a recurrent edge which feeds the hidden state from the previous time step to the hidden layer at the present time step. This process is repeated for each time step. The net input to the hidden layer is given by the equation:
Where $\boldsymbol{W_{xh}}$: the weight vector between the input layer and the hidden layer

$$\boldsymbol{z_h}^{(t)} = \boldsymbol{W_{xh}x_{(t)}} + \boldsymbol{W_{hh}h_{(t-1)}} + \boldsymbol{b_h} \quad (3.6)$$

$\boldsymbol{x_{(t)}}$: *input vector at time step t*

$\boldsymbol{W_{hh}}$: *weight matrix at the recurrent edge*

$\boldsymbol{h_{(t-1)}}$: *output of hidden state from the previous time step*

$\boldsymbol{b_h}$: *bias vector corresponding to the nodes in the hidden layer*

$\boldsymbol{z_h}^{(t)}$: *net input to hidden layer at time step t*

This net input is passed through the activation function in the hidden layer to get the hidden state at that time step. The activation function is usually a hyperbolic tangent (*tanh*) or sigmoid function (see Sec 3.3.1) [56, 67] .

The output of hidden layer or hidden state at a time step is given by:

$$\boldsymbol{h_{(t)}} = \phi_h(\boldsymbol{z_h}^{(t)}) \quad (3.7)$$

The output at time step t is given by :

$$\boldsymbol{y_{(t)}} = \phi_y(\boldsymbol{W_{ho}h_{(t)}} + \boldsymbol{b_o}) \quad (3.8)$$

Where $\boldsymbol{W_{ho}}$: *weight matrix between the hidden layer and output layer.*

$\boldsymbol{h_{(t)}}$: *output of hidden state at the time step = t, representing the current time*

$\boldsymbol{b_o}$: *bias vector corresponding to the output units.*

$\boldsymbol{y_{(t)}}$: *output vector at time step = t*

ϕ_y : *Activation function at the output layer*

ϕ_h : *Activation function at the hidden layer*

Based on the output values predicted by the model, the loss gets calculated by taking the difference between the true value and predicted value. The total loss is the sum of losses across all the time steps. The total loss is given by the equation :

$$L = \left(\sum_{t=1}^{t=n} L_{(t)} \right) \tag{3.9}$$

Where L : *total loss across all time steps or words in a sequence forming the sentence*

n : *number of words or time steps in a sequence*

$L_{(t)}$: *Loss at time step t*

The weights are updated in an RNN through the process of optimisation of loss through gradient descent algorithm by backpropagation through time (BPTT) [68]. Gradients are the rate at which the loss function changes with respect to weight, this process of calculating them is referred to as backpropagation. BPTT refers to backpropagation through time steps. The trainable parameters in a RNN are the weight matrices $\boldsymbol{W_{xh}}, \boldsymbol{W_{hh}}, \boldsymbol{W_{ho}}$.

The gradient of total loss is computed for each weight matrix.

$$\frac{dL}{d\boldsymbol{W_{xh}}} = \sum_{t=1}^{t=n} \frac{dL_{(t)}}{d\boldsymbol{W_{xh}}} \tag{3.10}$$

$$\frac{dL}{d\boldsymbol{W_{hh}}} = \sum_{t=1}^{t=n} \frac{dL_{(t)}}{d\boldsymbol{W_{hh}}} \tag{3.11}$$

$$\frac{dL}{d\boldsymbol{W_{ho}}} = \sum_{t=1}^{t=n} \frac{dL_{(t)}}{d\boldsymbol{W_{ho}}} \tag{3.12}$$

n : *number of words or time steps in a sequence*

The gradient of loss with respect to the weight matrix between the hidden layers are given by:

$$\frac{dL_{(t)}}{d\boldsymbol{W_{hh}}} = \frac{dL_{(t)}}{\boldsymbol{dy}_{(t)}} \times \frac{\boldsymbol{dy}_{(t)}}{\boldsymbol{dh}_{(t)}} \times \left(\sum_{k=1}^{t} \frac{\boldsymbol{dh}_{(t)}}{\boldsymbol{dh}_{(k)}} \times \frac{\boldsymbol{dh}_{(k)}}{d\boldsymbol{W_{hh}}} \right) \tag{3.13}$$

The gradient of loss with respect to the weight matrix between the hidden layer and outputlayer is given by:

$$\frac{dL_{(t)}}{d\boldsymbol{W_{ho}}} = \frac{dL_{(t)}}{\boldsymbol{dy}_{(t)}} \times \frac{\boldsymbol{dy}_{(t)}}{d\boldsymbol{W_{ho}}} \tag{3.14}$$

Where t : represents the present time step in a sequence of words

k : can take values between *1* to the present time step t

$\frac{\boldsymbol{dh}_{(t)}}{\boldsymbol{dh}_{(k)}}$ is calculated by multiplying adjacent time steps. So the gradient at a time step is dependent on the output of the hidden layer in the previous time step. As the number of time steps before the time step in focus increases, RNN finds it difficult to take into account the effect of those nodes as the derivatives of activation functions at those nodes becomes

negligibly small leading to a small weight update. This dependency leads to the problem of vanishing or exploding gradient problem as described below.

Major challenge faced by the RNN are :

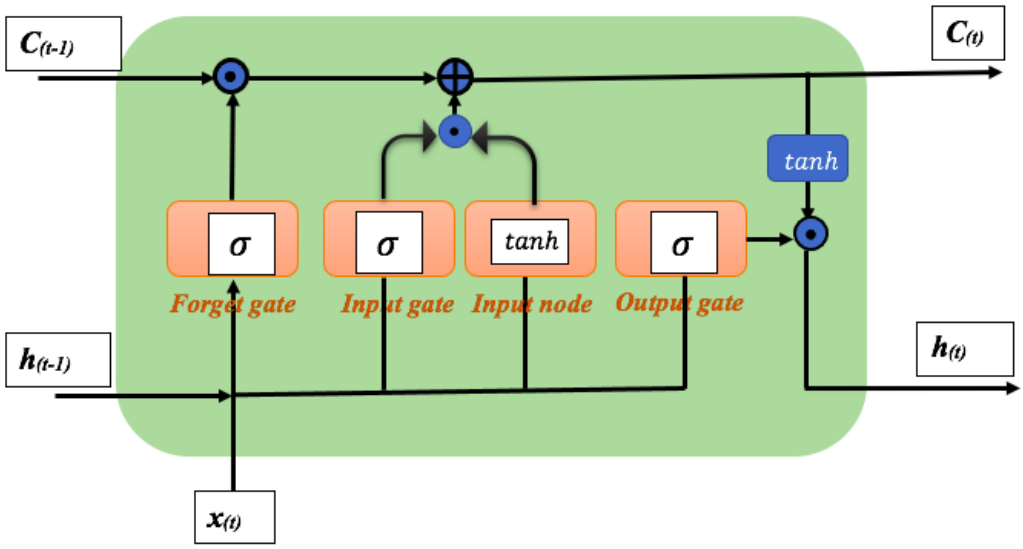
1. Vanishing or exploding gradient problem : The problem of vanishing gradient was first found and explored in 1991 by Sepp Hochreiter in his study "Fundamental Deep learning Problem". He found that the back-propagated error signals either shrink or grow rapidly making the weight updates become negligibly small or extremely large, preventing the model from reaching the global minima of loss leading to the vanishing and exploding gradient problem [59]. The type activation function in the hidden layer could lead to vanishing or exploding gradient problem. For example ,if the sigmoid function is the activation function used in the hidden layer the output will have values between *0* and *1*. Derivatives of the sigmoid activation function have small values between *0* and *0.25*. Multiplication of small values will result in a smaller value, thus making the weight update negligible, preventing the model from reaching the global minima, leading to the vanishing gradient problem. On the other hand if the activation function selected has a derivative greater than *1*. It will lead to an exploding gradient problem as the weight update is large thus preventing the model from reaching the global minima.
2. Captures information only from one direction: Since the hidden layer at each time step gets the input from the input layer at that time step and the hidden state from the previous time step, the past context is only captured. This suggests that when predicting the output for a specific time step the full context in which the word appears is not taken into consideration.

Long Short Term Memory (LSTM)

RNN fails to learn long-term dependencies as per the study conducted by Paul J Werbos [68] and suffers from vanishing and exploding gradient problem, as mentioned in Sec. 3.3.2. These challenges can be solved using an LSTM, which is a variation of a RNN, designed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber. The basic architecture of RNN and LSTM is the same with an input layer, hidden layer and an output layer. However, the only difference is in

the operations that occur inside the hidden layer. The building block of LSTM is a memory cell that replaces the hidden layer concept in an RNN.

Figure 3.16 represents the memory cell of an LSTM. The memory cell comprises of various components: the input gate, input node, forget gate and the output gate. There also exist different operators like element-wise multiplication and element-wise addition. It is essential to understand the concept of cell state and hidden state to understand the working of LSTM better. The hidden state $\boldsymbol{h_{(t)}}$ is the output from the memory cell at a point in time *t*, and the cell state $\boldsymbol{C_{(t)}}$ is the internal state which is the long term memory. Both these values are computed through a series of operation that takes place inside the memory cell. At a point in time *'t'*, the memory cell is fed with input $\boldsymbol{x_{(t)}}$ at that time step, the hidden state from the previous time step $\boldsymbol{h_{(t-1)}}$, and cell state from previous time step $\boldsymbol{C_{(t-1)}}$.



- $C_{(t-1)}$: *cell state of the previous time step*
- $h_{(t-1)}$: *hidden state of the previous time step*
- $C_{(t)}$: *cell state of the current time step*
- $h_{(t)}$: *hidden state of the current time step*
- $x_{(t)}$: *input at the current time step*
- ⊕ : *element wise addition*
- ⊙ : *element wise multiplication*

Figure 3.16: Architecture of LSTM memory cell

The hidden state representing the output from the previous time step $\boldsymbol{h_{(t-1)}}$ along with the input from the present time step $\boldsymbol{x_{(t)}}$ is fed to a forget gate which controls the process of forwarding the cell state from the previous time step $\boldsymbol{C_{(t-1)}}$. The forget gate has a sigmoid activation function that helps shrink the input fed to it to values between *0* and *1*. The output of the forget gate along with the cell state from the previous time step is fed to an element-wise multiplication operator that controls the cell state's flow from the previous time step, $\boldsymbol{C_{(t-1)}}$. If the value from the forget gate is *0*, it neglects the cell state signal, and if it is one, it will allow the signal to pass through. The current time step's cell state is also controlled by the signal coming from the input node and the input gate. The input gate has a sigmoid activation function while the input node has a hyperbolic tangent activation function. The input node computes the input, while the input gate controls the amount of input to be forwarded to the cell state $\boldsymbol{C_{(t)}}$. Both these signals from the input node and the input gate are fed to an element-wise multiplication operator, enabling the input gate to tune the amount of input signal to pass through.

A element-wise sum of previous cell state $\boldsymbol{C_{(t-1)}}$ forwarded after the element-wise multiplication with the output of forget gate and the signal forwarded after the element-wise multiplication of output of the input gate and input node is forwarded as the current time step's cell state $\boldsymbol{C_{(t)}}$. This signal is split into two copies where one copy flows as the cell state at the time step *"t"*, while the other copy is fed through a hyperbolic tangent to shrink the values between *-1* and *1* for the process of computing the hidden state at the time step *"t"*. $\boldsymbol{h_{(t)}}$which is the output at that particular time step is calculated by combining the previous step's output, which is the processed cell state, with the output from the output gate. The output gate has a sigmoid activation function, and this controls the flow of the cell state to the hidden state based on its value. If the value is *0*, it prevents the processed cell state from passing while if the value is one, it allows the processed cell state to pass completely. $\boldsymbol{C_{(t)}}$ and $\boldsymbol{h_{(t)}}$is then forwarded to the next time step [68].

Bi-directional Long Short Term Memory (Bi-LSTM)

We saw earlier that LSTM [68], a variation of an RNN manages to overcome the vanishing gradient problem faced by the RNN. Simultaneously, a Bi-directional LSTM is a modified version of LSTM that manages to overcome the challenge of capturing only prior information,

by taking into account the information from both directions [69] . The hidden layer comprises two LSTMs in parallel, one operating in the original order of sequence and the other in the reverse order of sequence.

A forward LSTM, represented by the dashed yellow bounding box in Figure 3.17 has information flow in the forward direction, representing the positive time direction or the orginal order of sequence. Consider a task of sequence labelling where a sentence composed of a sequence of tokens is fed as input to the neural network model. The forward LSTM is similar to the basic LSTM which captures the past information at each time step where the hidden layer at each time step receives as input the hidden state from the previous time step and the input at present step.

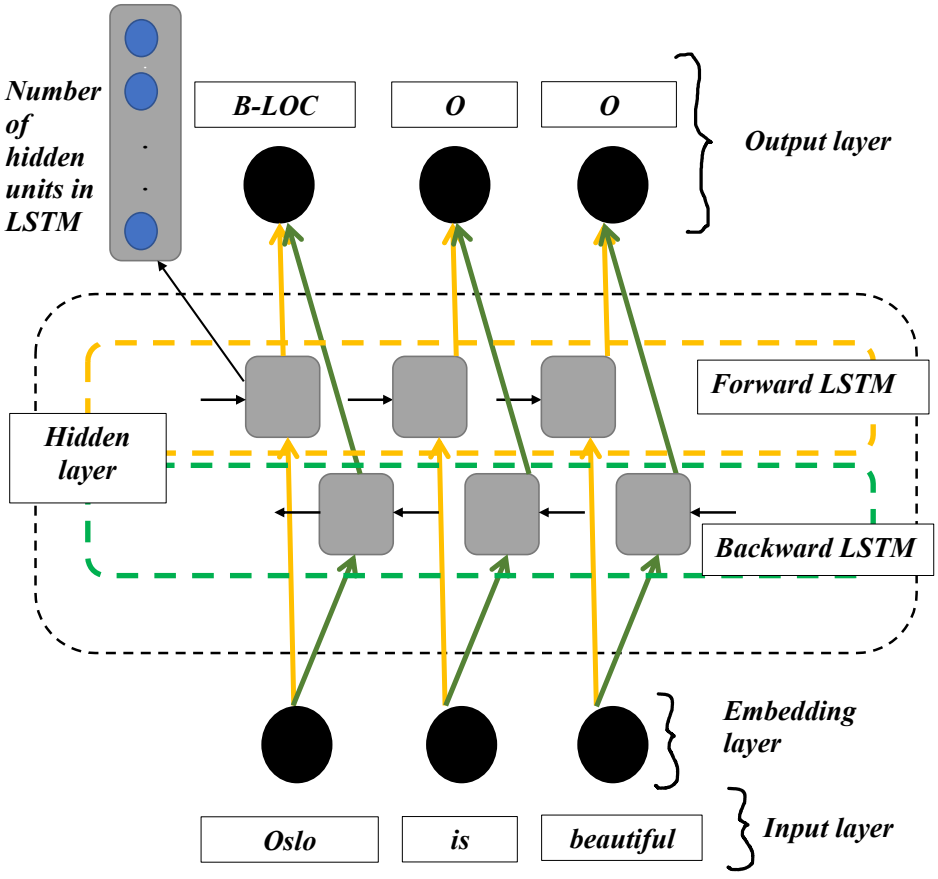


Figure 3.17: Architecture of Bi-LSTM

The other LSTM bounded by the green dashed box represents the backward LSTM, where the information flow begins from the end of the sentence, in the negative time direction or reverse order of sequence and the process takes place in the same manner as the forward LSTM. At each time step, the output of the hidden states of the forward LSTM and backward LSTMs are

concatenated to form the final output vector. The final output vector for each word is composed of scores known as emission scores corresponding to each label and the label with the highest score is selected as the predicted label for that particular word. To build a NER model we need use a classifier at the end of the Bi-LSTM layer. This could be a Softmax classifier [66] or a CRF classifier [70]. A SoftMax classifier is implemented using a Time distributed dense layer with Softmax activation function which can be used at the output layer to predict the word's label (see Sec. 3.3.1) [17].

3.3.3 Conditional Random Fields (CRF)

The output tags predicted by using a Bi-LSTM do not take into account the tagging constraints. Tagging constraints refers to the set of rules associated with the tagging scheme. Few possible constraints associated with the IOB2 tagging scheme are [71] :

- In a named entity composed of multiple words, the first word should begin with *B*-prefix tag and should not begin with *I*-prefix tag or *O* tag.
- If the tags are *B-label1*, *I-label2, I-label3* etc. *label1, label2* and *label 3* must be members of the same named entity.
- A word with *O* tag cannot be followed by *I* tag.
- In case of a named entity composed of multiple words, *B* tag should always be followed by *I* tag.

According to the IOB2 [25] tagging scheme *B* represents the beginning of a named entity, *I* represents the word that belongs to the inside part of the entity and *O* represents something which is not an entity or is outside an entity. A Bi-LSTM sequence labelling model does not consider the fact the first word in a sentence should begin with *B*-tag or *B*-prefix tag should always be followed by an *I*-tag (see Sec. 2.2). CRF manages to overcome this drawback and takes into account these constraints, thus ensuring that the predicted sequence of labels are valid [72, 73]. To predict the final label of the sequence of words the output vector of a Bi-LSTM is fed to a CRF layer. A CRF takes into account the neighbouring tag information before predicting the current tag [73]. The output of Bi-LSTM layer is in the form of vectors representing score for each label for a word. This score vector known as emission score is fed to the CRF layer where the sequence of tags with the highest prediction score is selected as the final prediction. If we have a sequence of observations, in a CRF we take into account the joint distribution of the labels with respect to the sequence of input tokens .In sequence labelling tasks, knowing the previous word would help in identifying the next word. In other words,

certain words are more likely to be followed by certain other words. CRF calculates the likelihood of all possible sequences and selects the sequence with the maximum likelihood. For example let x_1, x_2, x_3 denote the sequence of words in a sentences and y_1, y_2, y_3 denote the tags. In a CRF we calculate the $P(y_1, y_2, y_3|x_1, x_2, x_3)$ the sequence which gives the maximum probability is chosen as the final label. Each sentence is a collection of tokens and our aim is to find the most likely path. CRF creates a bunch of combination of paths and creates a transition score matrix for each path. This best path is decided based on the total score calculated using the transition score and emission score.

CRF is a conditional probability distribution model which assumes that the output random variables constitute Markov Random Fields. A conditional probability distribution model assumes that the input sequence and the output sequence are random variables. The model finds the optimal path of a sequence by computing the conditional probability of a sequence of words with respect to the sequence of tags . The path that gives the maximum probability is selected and fixed as the prediction. It uses the Viterbi algorithm, a dynamic programming algorithm, to find the optimum path [72, 73].

The process of finding the best sequence of labels for a input sequence is as follows, the output vectors produced by Bi-LSTM layer represents the emission score. Each word will have a vector representing the emission score for each tag, if *B-PER, I-PER, B-LOC, I-LOC* and *O* are the tags associated with a NER task . Then each word will have a *five-dimensional* vector with each value in the vector representing the emission score corresponding to the tag. The transition score is obtained from the transition matrix. It refers to the score associated with transforming from one label to the other including the start point and end point. This matrix is created by randomly initialising each score in the matrix and this value is automatically updated during the training process.

Consider an input sentence $X = [x_1, x_2, x_3, x_4,..]$ and sequence of labels $Y = [y_1,y_2,y_3,y_4,\ldots]$. $\boldsymbol{E}$ represent the emission score matrix of dimension $k \times n$, where k is the number of labels, n is the number of words in an input sentence. E_{ij} corresponds to the emission score of the j^{th} tag for the i^{th} word in the sentence.

T is the transition score matrix where T_{ij} represents the transition score for transitioning from i^{th} tag to the j^{th} tag. **T** is a square matrix of size $k + 2$ where the additional two tags are for

the start point and end point. Transition scores are computed also for transitioning from start tag and end tag to the other tags.

The total score $s(X, y)$ is calculated as:

$$s(X, y) = \sum_{i=0}^{n} T_{y_i, y_{i+1}} + \sum_{i=1}^{n} E_{i,j} \quad (3.15)$$

A softmax for all the tag sequences gives a probability for a sequence of labels *y* for an input sequence *X* :

$$P(y|X) = \frac{e^{s(X,y)}}{\sum_{y \in Y_x} e^{s(X,y)}} \quad (3.16)$$

During the process of training the log probability of correct tag sequence is maximised:

$$log\big(P(y|X)\big) = s(X|y) - log\left(\sum_{y \in Y_x} e^{s(X,y)}\right)$$

$$= s(X|y) - log\big(\sum_{y \in Y_x} s(X, y)\big) \quad (3.17)$$

Where Y_x : all possible path of tags corresponding to a input sequence, also includes the one's that do does satisfy IOB2 tagging constraints.

And thus by comparing each value we find the sequence of labels that gives the best total score for a sequence of inputs [16]:

$$y_{best} = argmax\ s(X, y) \quad (3.18)$$

3.3.4 CRF Loss

Loss function refers to a function used to evaluate an algorithm's efficiency in performing a particular task by calculating the difference between the predicted values for output and the actual values. The CRF loss function includes the real path score and total score of all possible paths. Real path refers to the actual path with the correct sequence of labels. Path score is calculated by taking the cumulative sum of emission score and transition score for the path. Score of the real path will be the highest among all paths possible. Suppose there are five labels *B-PER, I-PER, B-LOC, I-LOC* and *O* and each sentence has four words. There are several paths possible like $Path_1$, $Path_2$, $Path_3$, … $Path_n$:

- $Path_1$ = *START, B-PER, B-PER, B-LOC, END*
- $Path_2$ = *START, O, O, B-PER, O, END*

If $Path_2$ is the real path then out of all possible paths $Path_2$ will have the highest score.

Total path score is calculated by summing the score of all possible paths .

$$Path_{total} = Path_1 + Path_2 + Path_3 + \cdots + Path_n \quad (3.19)$$

$$\text{Loss} = \frac{(Actual\ path)}{Path_{total}} \quad (3.20)$$

While training a model, we aim to optimise the model's parameters, thus making them efficient. This optimisation could be minimising or maximising the value. During the training of a Bi-LSTM-CRF, we aim to minimise the loss by increasing the proportion of real path score among the total score of all possible paths [71].

3.3.5 Optimiser

Optimisation, in general, refers to the process of minimising or maximising a parameter. In this study, we aim to optimise the neural network by minimising the cost function, dependent on the prediction error. The terms loss function and cost function are used interchangeably but they have slight difference. Cost function refers to the average prediction error for the entire training data. The optimisation process is performed by the optimisers, activated during the learning process while training a model. Optimisers use gradient descent algorithm where the error is calculated based on the gradient of the loss function, and the error is reduced by making the model move in the opposite direction of the gradient to reach the minimum or optimum value of loss function by updating the weight. Each weight update by backpropagation is referred to an iteration.

The optimiser used in this study is Root Mean Square propagation (RMSprop) [74] which is an adaptive learning algorithm which performs optimisation by mini-batch stochastic gradient descent (see Sec. 3.4.2). An RMSprop adaptively adjusts the learning rate in a way that it does not become negligibly small and thus preventing weight updates from happening leading to the vanishing gradient problem [74]. Additionally, since the gradients are different for each mini-batch within an epoch, using the same learning rate may not be the most efficient approach. Hence the new learning rate is calculated at the end of each mini-batch with the help of a parameter known as exponential weighted average of squares of gradient of loss with respect to weight (*Sdw*) and exponential weighted average of squares of gradient of loss with respect to bias (*Sdb*).

During the calculation of *Sdw* and *Sdb*, β parameter is used to restrict the square of gradient of loss with respect to weight and square of gradient of loss with respect to bias from becoming exponentially large. *Sdw and Sdb* at the current mini-batch is calculated by the formula:

$$Sdw(t) = \beta * Sdw(t-1) + (1-\beta)\left(\frac{dL}{dw_t}^2\right) \quad (3.21)$$

$$Sdb(t) = \beta * Sdb(t-1) + (1-\beta)\left(\frac{dL}{db_t}^2\right) \quad (3.22)$$

Where $Sdw(t)$: *square of gradient of loss with respect new updated weight at the current iteration*

$Sdb(t)$: *square of gradient of loss with respect to new updated bias at the current iteration*

$Sdw(t-1)$: *square of gradient of loss for the old weight at the previous iteration*

$Sdb(t-1)$: *square of gradient of loss for the old bias at the previous iteration*

$\frac{dL}{dw_t}$: *gradient of loss with respect weight at current iteration*

$\frac{dL}{db_t}$: *gradient of loss with respect to bias at current iteration*

The computation of new learning rate during mini-batch gradient descent in a RMS prop optimiser is given by :

$$\eta'_w = \frac{\eta_w}{\sqrt{Sdw(t)+\varepsilon}} \quad (3.23)$$

Where η'_w : *new learning rate for weight update at current iteration*

η_w : *previous learning rate for weight during previous iteration*

$$\eta'_b = \frac{\eta_b}{\sqrt{Sdb(t)+\varepsilon}} \quad (3.24)$$

Where η'_b : *new learning rate for bias update at current iteration*

η_b : *previous learning rate for bias during previous iteration*

ε : *A very small positive value introduced to prevent the denominator from becoming 0*

Formula for updating the weight and bias:

$$w_t = w_{t-1} - \eta'_w * \frac{dL}{dw_{t-1}} \quad (3.25)$$

$$b_t = b_{t-1} - \eta'_b * \frac{dL}{db_{t-1}} \quad (3.26)$$

Where

b_t : *new bias or updated bias at current iteration*

b_{t-1} : *old bias value or previous bias at previous iteration*

w_t : *new weight or updated weight at current iteration*

w_{t-1} : *previous weight or old weight at previous iteration*

L : *Loss calculated at the end of current iteration*

$\frac{dL}{dw_{t-1}}$: *gradient of loss with respect weight at previous iteration*

$\frac{dL}{db_{t-1}}$: *gradient of loss with respect to bias at the previous iteration*

Since the gradient is divided by square root of the average of gradients the optimiser has the name root mean square propagation. The beta value restricts the *Mean square* from becoming exponentially big and thus preventing the new learning rate from becoming negligibly small. This makes it possible to compute learning rates that lead to optimal changes in the weights thus enabling the model to reach the global minimum. There exist an RMSprop version with momentum [75]. By default the momentum of RMSprop is *0.0* and Hinton suggested β value to be set as *0.9* and a good default initial learning rate (η) to be *0.001* while using a RMS prop optimiser.

3.3.6 Word vectorisation

In a NER task a sequence of words representing a sentence is fed as input to the neural network. The complete input to the network comprises of multiple such sentences. NLP tasks using deep learning algorithms can only work with numeric vectors and not characters or words. Textual representation of data does not provide information about the relationship between the words. Hence, these words have to be converted to vectors, a numerical representation of the words in *multi-dimensional* space known as word embeddings. Representing them in vector format makes it possible to perform arithmetic operations and thus get relationship between words. It

also makes it possible to group words based on their orientation in the *multi-dimensional* space. Hence, during the stage of pre-processing textual data before feeding it to the deep learning network, the words are converted to vectors of real numbers by a technique known as word vectorisation [35]. Pre-processing of the data for making it suitable for performing word vectorisation includes tokenising and padding as explained in detail in Sec. 3.2. There exist several ways to convert words to input features or vectors. In this thesis, we are focusing on three major techniques listed below:

1. Word vectorisation by embedding layer.
2. Word vectorisation by pre-trained word embeddings.
3. Word vectorisation by contextual word embeddings.

Word vectorisation by Embedding Layer

One-hot encoding is one of the naive ways for performing the task of converting words to input features. In one-hot encoding, the sequence of words are converted to unique indices of the words in the data set (see Sec. 3.2.2) , and then each word is converted to a vector of only zeroes and ones. The vector generated for each word is sparse, because of many zeroes and is high dimensional. The dimension of each word vector corresponds to the unique number of words in the data set, leading to an increase in the word vector dimension as the vocabulary increases. Thus, the models trained using one-hot encoding suffer from the "curse of dimensionality". This led to the development of an approach of using the embedding layer for generating a dense, finite-dimensional vector representing input features. The dimension of the vector generated is a parameter that can be specified based on our preference. During instantiation, the initial vectors are randomly instantiated, and these are adjusted during the training of the model by the process of backpropagation [68]. This representation of words can have a dimension smaller than the number of unique words in the data set [55]. The Figure 3.18 depicts an illustration of generating vectors corresponding to each word. As discussed in the preprocessing part in Sec. 3.2.2, during the process of tokeninsing the words are converted to unique indices.

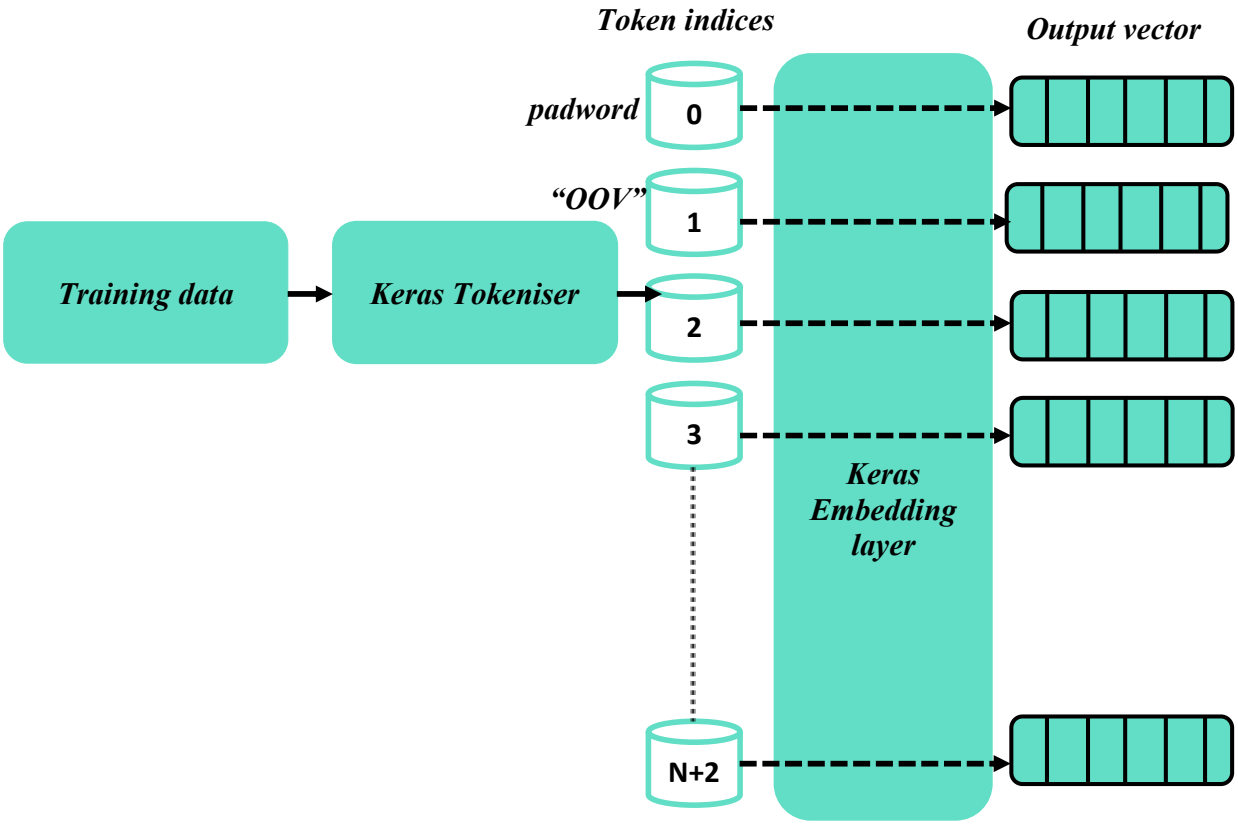


Figure 3.18: Training Keras tokeniser on the training data, followed by word vectorisation by Keras embedding layer, where 0 index is reserved for pad words and 1 index for "OOV": out of vocabulary words

There exist *"N+2"* token indices as in Figure 3.18, where *N* represents the number of unique words in the data set. The two additional indices *0* and *1* are reserved for the word used for padding the sequence and the other for the unknown words or words which are outside the vocabulary set of the training data. Each row in the trainable weight matrix corresponds to the vector associated with each word at the unique index. The resulting weight matrix of the embedding layer is of dimension *(N+2 × output dimension)*. The output dimension is the pre-defined dimension for the output vector which could be defined while building the model. The embedding layer can be implemented using TensorFlow Keras. Figure 3.19 shows the code used to create a model and add an embedding layer and Figure 3.20 represents the model architecture of Bi-LSTM-CRF NER using Keras embedding layer for word vectorization.

```
from keras.models import Model, Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
from keras_contrib.layers import CRF
def build_model():

# Model definition
    input = Input(shape=(MAX_LEN,)) # number of columns in the input , tells the model how much data to expect

    model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & OOV)
                      input_length=MAX_LEN, mask_zero=True)(input)

    model = Bidirectional(LSTM(units=50, return_sequences=True,
                               recurrent_dropout=0.1, kernel_initializer=keras.initializers.glorot_uniform(seed=66)))(model)

    model = TimeDistributed(Dense(50, activation="relu", kernel_initializer=keras.initializers.glorot_uniform(seed=66)))

    crf = CRF(n_tags+1)  # CRF layer, n_tags

    out = crf(model)  # output

    model = Model(input, out)

    model.compile(optimizer="rmsprop", loss=crf.loss_function, metrics=[crf.accuracy])

    model.summary()
    return model
```

Figure 3.19: Embedding layer definition for random initialisation of embeddings by Keras embedding layer

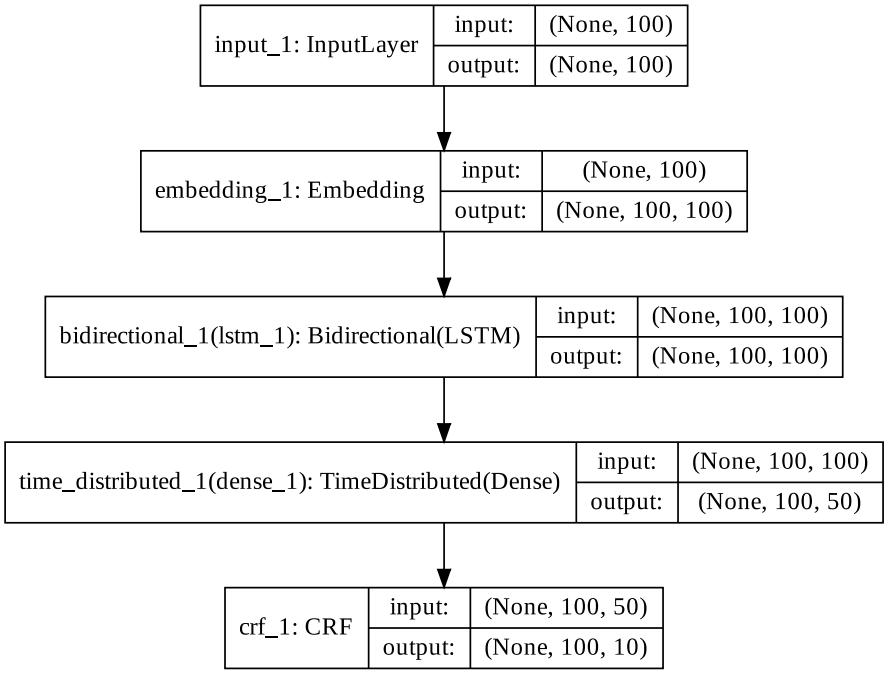


Figure 3.20: Architecture of Bi-LSTM-CRF using word vectorisation by Keras embedding layer

Word vectorisation by pre-trained word embeddings

When the size of the training data set is small. The process of learning embeddings from the data becomes challenging. This is challenging due to two significant reasons [76]:

1. **Sparse training data:** Most real-world data sets have many rare words. This triggers the need for a data set that is rich in vocabulary to make the training result in an embedding vector that is representative of the word.
2. **The number of trainable parameters:** Training a model from scratch is a time-consuming process because of the enormous number of trainable parameters.

To handle these challenges, we use pre-trained word embeddings. Pre-trained word embeddings use the concept of transfer learning. It is a mechanism in machine learning where the output or learned parameters from one task are used as input in another task, thus transferring the results from one experiment to another. In pre-trained word embeddings, the parameter getting transferred are word vectors, also referred to as word embeddings. Instead of initialising the word vectors by random a vector and then training them to create embeddings for words in a data set, we use these pre-trained embeddings. The data set on which these embeddings are trained is large, thus making it possible to effectively capture the syntactic and semantic meaning of a word and this information is saved as weights and can be used in other tasks [76]. The Keras embedding layer makes it possible to initialise the weights of the embedding layer with the weight matrix from the pre-trained model during the process of building the NER system.

```
def build_glove_model():

    from keras.models import Model, Input
    from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
    from keras_contrib.layers import CRF


    input = Input(shape=(max_len,))

    model = Embedding(input_dim = num_words, output_dim = max_len, input_length = max_len,weights=[embedding_matrix], trainable = False, mask_zero= True)(input)

    model = Bidirectional(LSTM(units=50, return_sequences=True,
                               recurrent_dropout=0.1, kernel_initializer=keras.initializers.glorot_uniform(seed=66)))(model)  # variational biLSTM
    model = TimeDistributed(Dense(50, activation="relu", kernel_initializer=keras.initializers.glorot_uniform(seed=66)))(model)  # a dense layer as suggested by n

    crf = CRF(n_tags+1)  # CRF layer, n_tags+1(PAD) to match to_categorical

    out = crf(model)  # output

    model = Model(input, out)

    model.compile(optimizer="rmsprop", loss=crf.loss_function, metrics=[crf.accuracy])

    model.summary()

    return model
```

Figure 3.21: Embedding layer definition using pre-trained GloVe word embeddings

In this study, we are focusing on pre-trained GloVe embeddings [77]. In 2014, as per the research results conducted by researchers of Stanford [77] GloVe outperforms all other models on multiple data sets. Hence they are best suited for subtasks of NLP tasks like named entity recognition. Figure 3.21 shows the model definition and model architecture of a Bi-LSTM-CRF NER using word vectorisation by pre-trained GloVe embeddings is similar to Figure 3.20 [77].

GloVe is a word vectorisation algorithm developed by researchers at Stanford. Four GloVe pre-trained models are available for download from [77] and can be used in the Keras embedding layer. The variation is in the dimension of the word vectors. They come as *50, 100, 200, 300*-dimensional vectors trained on Wikipedia 2014 data and English Giga word fifth edition newswire text data with *6* billion tokens and *400,000*-word vocabulary. In this study, we are using the *100-dimensional* word embeddings version. These are pre-trained word embeddings encapsulating the vector representation of words that are learned by unsupervised learning. GloVe embeddings are generated by training on the non-zero entries of a global word-to-word co-occurrence matrix. This matrix holds information about relationship between words, how often a pair of words occur together in a context. This is calculated on the basis of the how relevant a word is to another word. This matrix is then factorised to obtain a low dimensional representation. The semantic relationship between the words are measured by calculating a metric representing the probability of two words occurring together [77].

Table 3.1 below illustrates the co-occurrence matrix for obtaining the semantic relationship between words for the sentence: "*notice of general meeting sent to shareholders*". The co-occurrence matrix can be considered as a count matrix which holds the number of times a pair of words appear together in the context. The size of the context window can be pre-defined . In this example a context window of one is chosen which takes into account one word on the left and right of the word in focus.

Table 3.1: The co-occurrence matrix for the sentence " notice of general meeting send to shareholders" with a window size of 1

	notice	***of***	***general***	***meeting***	***sent***	***to***	***shareholders***
notice	0	1	0	0	0	0	0
of	1	0	1	0	0	0	0
general	0	0	0	1	0	0	0
meeting	0	0	1	0	1	0	0
sent	0	0	0	1	0	1	0
to	0	0	0	0	1	0	1
shareholders	0	0	0	0	0	1	0

From the co-occurrence matrix we calculate the probability of a pair of words e.g. the word "*meeting*" to find which word is most relevant to the word "*meeting*" :

$$\mathcal{P}\left(\frac{"meeting"}{"general"}\right) = \frac{number\ of\ times\ the\ word\ "meeting"\ appears\ with\ the\ word\ "general"\ in\ the\ sentence}{the\ number\ of\ time\ the\ word\ "meeting"\ appears\ in\ the\ sentence} \quad (3.27)$$

$$number\ of\ times\ the\ word\ "meeting"\ appears\ with\ the\ word\ "general"\ in\ the\ sentence\ = 1. \quad (3.28)$$

$$number\ of\ times\ the\ word\ "meeting"\ appears\ in\ the\ sentence\ = 1. \quad (3.29)$$

$$\mathcal{P}\left(\frac{"meeting"}{"general"}\right) = \frac{1}{1} = 1 \quad (3.30)$$

$$\mathcal{P}\left(\frac{"meeting"}{"sent"}\right) = \frac{number\ of\ times\ the\ word\ "meeting"\ appears\ with\ the\ word\ "sent"\ in\ the\ sentence}{the\ number\ of\ time\ the\ word\ "meeting"\ appears\ in\ the\ sentence} \quad (3.31)$$

$$number\ of\ times\ the\ word\ "meeting"\ appears\ with\ the\ word\ "sent"\ in\ the\ sentence\ = 1. \quad (3.32)$$

$$\mathcal{P}\left(\frac{"meeting"}{"sent"}\right) = \frac{1}{1} = 1 \quad (3.33)$$

To find which word is more relevant to the word in focus , here "*meeting*" among the words "*sent*" and "*general*", we take the ratio of *(3.30* and *(3.33*. If the ratio is > *1* it means the top word is more relevant than bottom word. If the ratio is close to *1*, it means both words are relevant to the word in focus [76]. In the above example both the words "general" and "sent"

are equally important to the word “meeting”. This concept of computing probabilities from the co-occurrence matrix is the main idea behind the algorithm used for generating GloVe embeddings. This algorithm is trained on non-zero values of co-occurrence matrix thus making the training much faster as the number of non-zero matrix values are much smaller in comparison with the total number of words in the corpus. The objective of the learning algorithm is to learn the word vectors in a way that their dot product of the word vectors is equal to the logarithm of the word’s probability of co-occurrence [77] .

In addition, GloVe embeddings considers both local as well as global statistics in terms of local context window and global matrix factorisation. Local context window in terms of the window size which decides the number of words to be considered on either side of the word in focus. Global statistics due to the fact that GloVe embeddings take into account the whole data corpus which includes multiple sentences in the case of NER task.

Word vectorisation by contextual word embeddings.

NER performed using pre-trained word embeddings does not consider the words' contextual information; instead, it focuses on the lexical representation. This leads to low performance of NER tasks performed using pre-trained embeddings when unfamiliar words out of the vocabulary appear in the data set. This led to contextual word embeddings [13], derived by using unsupervised language models to predict a word's representation in each context. These succeed in addressing the challenges of using pre-trained word embeddings listed below [13] :

1. To make the model capable of learning the syntactical and semantical characteristics of words. Syntactical characteristics take into account the grammatical structure of the words.
2. Model the word polysemy, which is associated with the existence of multiple meanings for a single word.

In [78], the authors conducted experiments to compare NER models' performance using contextual word embeddings, and it was concluded that the Embeddings from Language Models (ELMo) embeddings gives stable results when working with words which are outside the vocabulary of the training data set. Hence in this thesis, we experiment with ELMo embeddings as an example for contextual embeddings for performing a comparative study of different word embeddings in performing NER tasks.

ELMo is Natural Language Processing (NLP) framework developed by AllenNLP [13]. The Figure 3.22 shows the ELMo architecture for generating word embeddings.

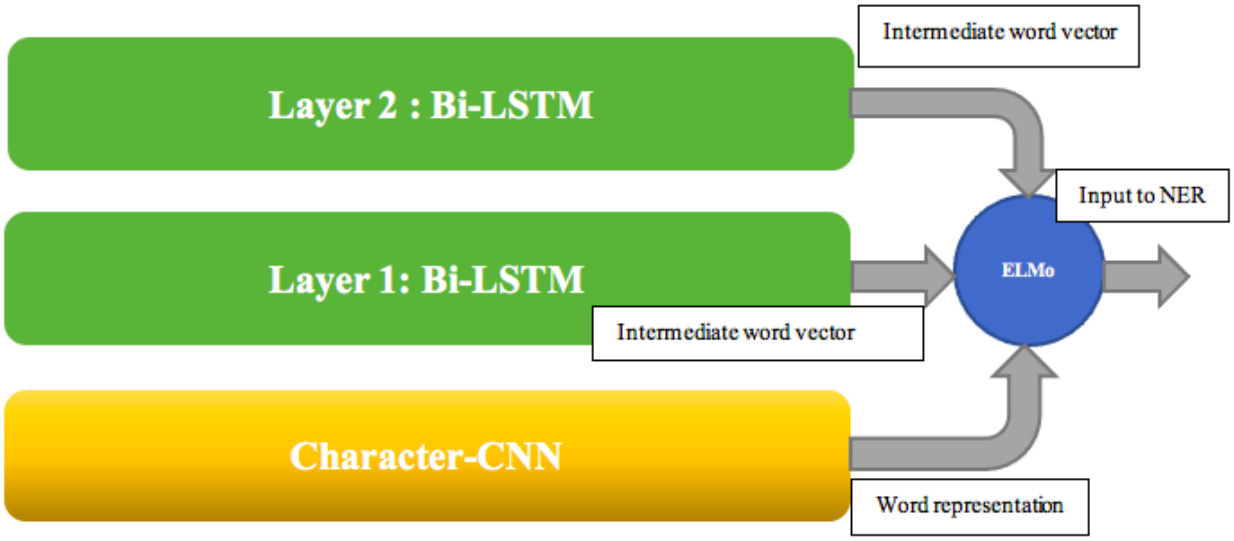


Figure 3.22: ELMo architecture

The ELMo architecture includes two layers of Bi-directional LSTM (Bi-LSTM) coupled with character-level Convolutional Neural Network (CNN), hence the name Embeddings from Language Models (see Figure 3.22). Unlike the traditional pre-trained embeddings, ELMo embeddings are dynamic and depend on the context in which the word occurs. These word representations are character-based with the help of a CNN, thus making the model capable of learning the word's morphology and generating valid representations for words outside the vocabulary.

Each layer of Bi-LSTM generates intermediate word vectors. The intermediate word vectors are generated by combining the word vector from the forward pass of the language model and the language model's backwards pass. The word vector generated during the forward pass of the model has information about the words before the word in focus, and the intermediate vector generated by the backward pass has information about the word itself and the words after it. Both these intermediate vectors aids in capturing the context in which a word appears. These intermediate vectors for the word are combined with the word representation from the Character-CNN to get the final representation which is fed as input to a NER model.

There are two possibilities to use an ELMo model for generating word embedding. We can either use a pre-trained ELMo model or train the ELMo model from scratch. This thesis uses the transfer learning approach of using the pre-trained ELMo model trained on 1 billion word benchmark using the model described in [13] - This can be downloaded from the TensorFlow hub.

```
elmo_model = hub.Module("https://tfhub.dev/google/elmo/2", trainable=True)
sess.run(tf.global_variables_initializer())
sess.run(tf.tables_initializer())
```

Figure 3.23:Loading pre-trained ELMo model from TensorFlow hub using python

```
def build_elmo_model():

    from keras.models import Model, Input
    from keras.layers.merge import add
    from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional, Lambda
    from keras_contrib.layers import CRF
    import tensorflow as tf

    os.environ['PYTHONHASHSEED']='0'
    np.random.seed(2)
    random.seed(2)
    tf.set_random_seed(2)

    input_text = Input(shape=(max_len,), dtype=tf.string)

    embedding = Lambda(ElmoEmbedding, output_shape=(max_len, 1024))(input_text)

    model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(embedding)

    model = TimeDistributed(Dense(50, activation="relu"))(model)

    crf = CRF(n_tags+1)

    out = crf(model)

    model = Model(input_text, out)

    model.compile(optimizer="rmsprop", loss=crf.loss_function, metrics=[crf.accuracy])

    return model
```

Figure 3.24: Lambda layer definition using ELMo embeddings

The Figure 3.23 shows the piece of code used to load the pre-trained ELMo model from TensorFlow hub. By setting trainable = *True*, it makes it possible to finetune some parameters, four trainable scalar weights for integrating the layers as described in the paper [13]. Figure 3.24 shows the piece of code in python used to build the model with ELMo embeddings. Figure 3.25 shows the architecture of Bi-LSTM-CRF with word vectorisation by ELMo embeddings.

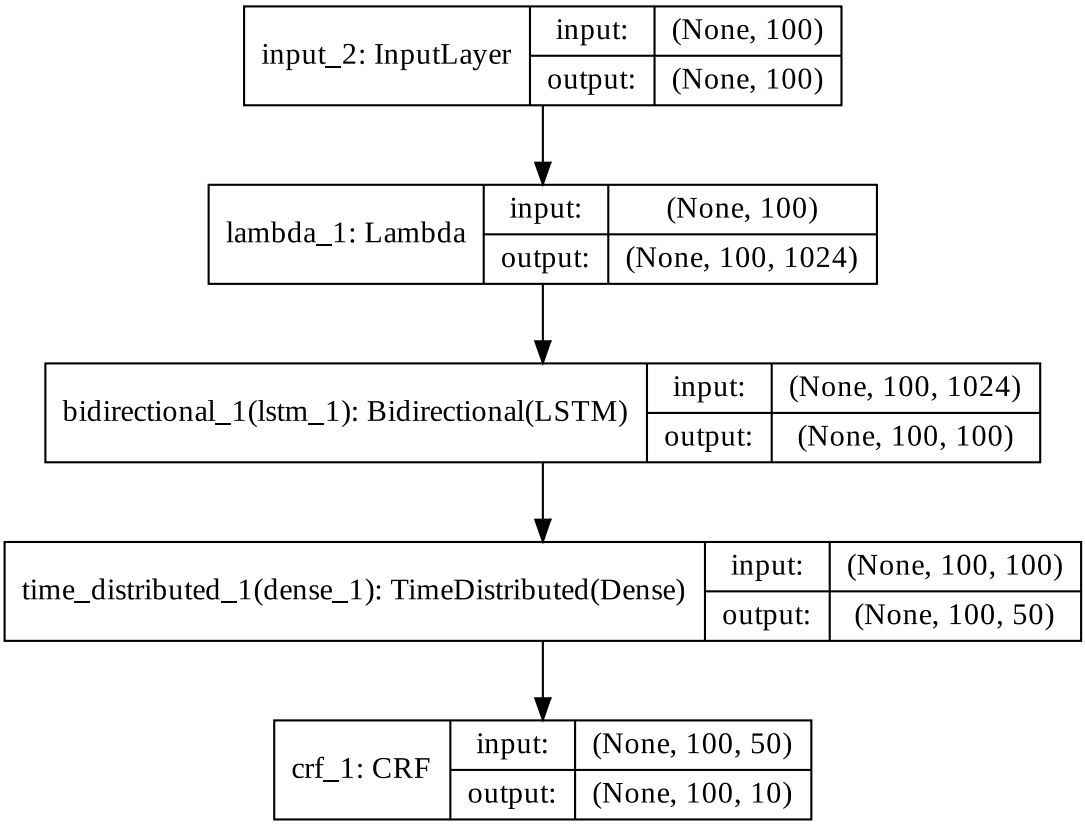


Figure 3.25: Architecture of Bi-LSTM-CRF with ELMo embeddings

3.3.7 Common model architecture

The deep learning models used for the comparative study of NER with different word vectorisation techniques consist of an input layer, embedding layer or lambda layer, hidden layers and an output layer (see Figure 3.17). The embedding layer varies depending on the type of word vectorisation used. In case of contextual word embeddings a lambda layer is used instead. A group of sentences are given as input to the model. Each sentence consists of *100* words as discussed in Sec. 3.2. Each word in a sentence is transformed to a vector (see Sec. 3.3.6) by the embedding layer or the lambda layer. The embedding corresponding to the words in the input sequence are then fed to the hidden layers, where they are processed. The processed output of the hidden layers are then fed to output layer which results in the prediction of label corresponding to each word in the input sentence.

Input Layer : Input layer takes as input a tuple containing the input dimension. The number of units in input layer thus corresponds to the dimension of the input data in this case *100* is the selected dimension or maximum length of sentences in the input data.

Embedding layer : The word vectorisation methods discussed in Sec. 3.3.6 is implemented in this layer. Based on the type of vectorisation technique used there comes variations in the definition of the embedding layer. The embedding layer makes it possible to learn the word vectors corresponding to a word by being part of a deep learning model. It also makes it possible to load pre-trained word embeddings. The embedding layer in general converts the words present in the data set to vector representation the dimension of the output vector is given as the input to the embedding layer. The actual orientation of the word vectors in the *multi*-dimensional space is learned during training the model. The embedding layer takes a 2D tensor in the form (*input dimension, input length*) as input. The output of embedding layer is a 3D tensor of dimension (*input dimension* × *input length* × *output dimension*). Corresponding to each word a vector of specified dimension gets generated during the process.

It takes *six* parameters :

1. **Input dimension**: Denotes the vocabulary size which includes the total number of unique words + two where two denotes the word used for padding and for word outside the vocabulary or unknown words.
2. **Output dimension** : Dimension of the embeddings.
3. **Input length** : Specified maximum length selected for padding, to transform all sequences to the same length.
4. **Mask zero** : This should be set as True or False depending on the problem. It is set to *True* as suggested by Keras TensorFlow documentation to notify the model that the input value contains padding and should be ignored from processing [79].
5. **Weights** : this attribute can be initialised with a weight matrix, this is used to load the pre-trained weight generated by pre-trained word embeddings such as GloVe embeddings
6. **Trainable** : This attribute takes Boolean values as input. It is used with pre-trained word embeddings where if this attribute is set to false the model remembers the weight matrix fed from the pre-trained model. If *trainable* is set *to True* the model gets

initialised with the pre-trained weights and during the training this weight changes. On the other hand, the attribute *trainable* is set to "*False*" to avoid forgetting the embeddings learned from the pre-trained model.

Lambda Layer: This is used instead of the embedding layer during word vectorisation by contextual word embeddings (see Sec. 3.3.6). The custom operations we are interested in performing are not supported by other Keras layers and that is where the Keras lambda layer makes it possible to do an operation on an input tensor and combine it to other layers of the model. Keras lambda layer is used load the pre-trained ELMo model from TensorFlow hub and generate word embeddings corresponding to the words in the input sequence and integrate with the Keras model. The Keras Lambda layer transforms each word in the sentence to a word feature of *1024* dimension, which is the default dimension of output vectors generated by pre-trained ELMo model [80].

Hidden Layer : Bi-LSTM layer and time distributed dense layer is implemented as the hidden layers in the model.

Bi-LSTM Layer : It takes four parameters as input:

1. **Units :** the number of neurons in the Bi-LSTM layer, it denotes the dimension of output from that layer from the Bi-LSTM.
2. **Return sequences :** If *return sequence* is *True*, it will return the full sequence of output ,otherwise it if the *return sequence* is *False*, it will return only the last output from the sequence of outputs.
3. **Recurrent dropout :** Fraction of the units to drop for the linear transformation of the recurrent state. It lies between *0* and *1*.
4. **Kernel initialiser :** Neural network need to be initialised with some weights at the beginning. A kernel initialiser defines how to set the initialise weights it has an attribute "seed" to produce the same random tensor . For this study we chose glorot uniform class initialised which draws samples from a uniform distribution between -*limit* and +*limit* where *limit = sqrt(6/number of input units in the weight tensor + number of output units in the weight tensor)* to fix the randomness in

initialising weight we fixed the seed of glorot uniform distribution to a value of 66 [81].

Time distributed dense layer: A fully connected dense layer refers to a layer where each neuron in the input is connected to every neuron in the dense layer [82]. A time distributed dense layer is a variant of a fully connected dense layer with a difference being the same fully connected dense layer is applied to the output of every time step from the previous layer. For each output of every time step of the previous layer is connected to every neuron in the dense layer. This type of layer is used for sequence labelling task which makes it possible to apply the same dense layer to each time step of an LSTM . It takes as input the number of nodes in the dense layer and the activation function. A time distributed dense layer with ReLU activation function (see Sec. 3.3.1) is used in this study. The reason for using ReLU activation function in the hidden layer is due to the fact that it does not activate all the neurons at the same time thus making the process computationally effectively . This also prevents the Vanishing or exploding gradient problem [59, 68].

Output layer: The CRF layer described in Sec. 3.3.3 is used as the output layer in the model. It takes as input number of tags.

3.4 Training, evaluation and comparison of results

This section explains the steps involved in performing the experiments on the data set and reproducing the results. The metrics and cross-validation technique used for evaluating the model performance are also explained along with the software used for setting up the environment for running the experiments.

3.4.1 Software used

The experiments were done on Google Colab [50] because of the availability of a larger RAM size of 12GB and a hosted runtime with GPU and TPU availability. Python deep learning libraries TensorFlow, and Keras API [79] were used for building the models. The versions of the packages used and the git hub link and git hash associated with the final code used for the experiment are encapsulated in tables (Table 3.3 and Table 3.3).

*Table 3.2: Versions of packages used in the experiments, where *Keras: word vectorisation by Keras embedding layer, *GloVe: word vectorisation by GloVe embeddings, *ELMo: word vectorisation by ELMo embeddings*

Software	Version	
	Bi-LSTM-CRF (*Keras, *GloVe)	**Bi-LSTM-CRF (*ELMo)**
python	3.7.10	3.7.10
pandas	1.1.5	1.1.5
Keras	2.2.4	2.2.4
Keras-contrib	2.0.8	2.0.8
Seqeval	1.2.2	1.2.2
Keras application	1.0.8	1.0.8
Keras pre-processing	1.0.9	1.0.9
NumPy	1.19.5	1.16.3
TensorFlow estimator	1.14.0	1.15.0
TensorFlow	1.14.0	1.15.1
scikit-learn	0.22.2.post1	0.22.2.post1
ELMo embeddings [83]	-	V2

Table 3.3: Versions of files used for this thesis is available in the GitHub repository: https://github.com/meerajsph/Master_thesis.git

Filename	Git hash
Data_analysis.ipynb	61a4d00
Bi-LSTM-CRF-NER-Keras.ipynb	847867b
Bi-LSTM-CRF-NER-GloVe100.ipynb	e4fd4fe
Bi-LSTM-CRF-NER-ELMo.ipynb	da94b22
Hyperparameter_tuning_Bi-LSTM-CRF-NER-GloVe100.ipynb	4bf72c8
Bi-LSTM-CRF-NER-GloVe50.ipynb	d9f73d7

3.4.2 Data handling by Mini batch gradient descent

Gradient descent [68] is an optimisation algorithm used by the neural models for learning iteratively (Sec. 3.3.5). Sample refer to a single row in a data set which is denoted by a sentence. The whole data set is composed of multiple samples, and each sample is composed of a list of tokens, representing the word and list of tags, representing the labels corresponding to each word. A batch, on the other hand, is composed of multiple samples. Depending on the number of samples in a batch the learning algorithm can be divided into three types. Firstly, if the number of samples in a batch is equal to the number of samples in the entire training data, it is known as batch gradient descent. Secondly, if the number of samples in a batch is one, then it is known as stochastic gradient descent, and finally, if the number of samples in a batch is greater than one and less than the number of samples in the training data, it is known as mini-

batch gradient descent. In this study we have selected a batch size of *32* and hence it is mini-batch gradient descent. The error is calculated at the end of a single batch. Epochs refer to the number of times you train the model on the entire training data. A single epoch refers to looping over the whole training data. Iterations refer to the number of times weight update occurs by back propagation within an epoch. The number of samples in a batch and the number of epochs are hyperparameters that can be adjusted to improve the model performance and make them optimal. Figure 3.26 shows mini-batch gradient descent for a single epoch where each green arrow down the curve denote an iteration by each batch within the single epoch. For an epoch the weight update occurs at the end of each batch. Weight gets updated for a batch within the epoch during backpropagation by *(3.34*.

$$w_t = w_{t-1} - \eta' * \frac{dL}{dw_{t-1}} \qquad (3.34)$$

where w_t : *New weight updated by backpropagation at current iteration*

w_{t-1} : *Old weight or previous weight at pervious iteration*

L : *Loss calculated during forward pass for the current iteration*

η' : *new learning rate, computed by the optimiser in use*

Learning rate decides the step size taken to reach the global minima during each weight update, represented by a single green arrow in the figure. Step size is represented by length of the green arrow.

Consider an example of model with *100* training samples with a batch size of *10* samples training for *5* epochs. It means the entire data set has been divided into *10* batches, and the training happens *5* times. During each epoch the entire training data gets trained, since we have a batch size of *10*, it divides the whole training data into *10* batches. So during the first epoch, the model is trained with *10* samples for *10* times, taking each batch at a time and this is repeated for rest of the four epochs. Forward propagation for calculating the loss and backward propagation for updating the weight occurs at the end of each batch of *10* samples within an epoch. The weight update at the end of each batch within an epoch is added to update the final weight at end of each epoch [84].

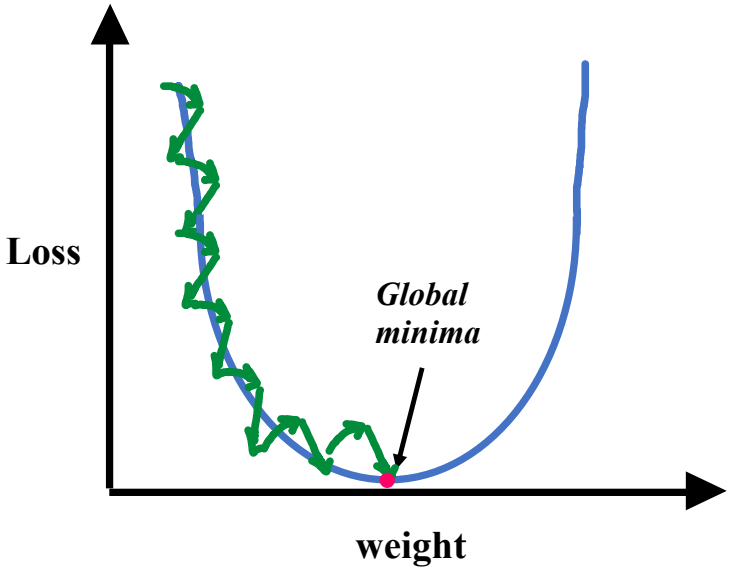


Figure 3.26: Mini batch gradient descent for weight update, where a green arrow represents the weight update during an iteration for a single mini batch and the red dot represents the Global minima which is the point where the loss is minimum

3.4.3 Evaluation Metrics

There exist various evaluation metrics for measuring the performance of NER systems in terms of its ability to perform classification tasks. In a survey conducted on the NER systems between 1991 and 2006 the evaluation metrics used by various forums, namely MUC, IREX, CoNLL and ACE conferences for evaluating the performance of NER during that period was reported [85]. The process of evaluating the NER systems by the forums were based on their ability to extract and label words. This was achieved by comparing the system's output to a human-created standard dataset with the correct tag corresponding to each word. It was highlighted that there exist variations in evaluation criteria in terms that some evaluation forums conducted evaluation based on partial matching, while others followed a strict matching system. In the strict mode of evaluation, an evaluation is considered correct only if the NER system manages to correctly find the entity boundaries and the class type to which it belongs. On the other hand, in partial matching, the evaluation is considered correct if a major part of the entity is correctly found and labelled as the correct class, in other words in partial matching identification of entity boundaries is not important.

In this study precision, recall, F1 score and accuracy are used for evaluating the model performance and strict mode evaluation criteria is used which is similar to the evaluation

criteria used in the CoNLL 2003 shared task [20]. Before explaining each evaluation metric it is important to understand the terms related to them. In a classification task, these terms are defined on a class level, which means taking a single class at a time.

***True positive* (*tp*):** A prediction of a class for a named entity is said to be a true positive if the predicted class is same as the actual class.

***True negative* (*tn*):** A prediction of a class for a named entity is said to be a true negative if they are correctly identified as a negative prediction or not belonging to the class when they actually do not belong to the class in focus.

***False positive* (*fp*):** A prediction of a class for a named entity is said to be a false positive if they wrongly classified to belong to the class in focus when they are actually not members of that particular class.

***False negative* (*fn*):** A prediction of a class for a named entity is said to be a false negative if a named entity actually belonging to the class in focus is wrongly classified as not belonging to the class.

Table 3.4: Evaluation metrics with their formula and definition

Metrics	Formula	Definition
Precision	$\frac{tp}{tp + fp}$	Precision of classifying named entities to belong to a class refers to the ratio of the number of entities correctly identified to belong to a class out of the total number of entities found to belong to a class by the NER system.
Recall	$\frac{tp}{tp + fn}$	Recall also known as Sensitivity, refers to the ratio of number of entities correctly found by the NER system out of the total number of entities in the data set.
F1-score	$\frac{2 \times Precision \times Recall}{Precision + Recall}$	F1-score is a metric used to summarise the performance of a model based on the precision and recall. This is done by calculating the harmonic mean. Hence the value of F1-score will always be between the precision and recall.
Accuracy	$\frac{tp + tn}{tp + fp + fn + tn}$	Accuracy measures the percentage of correct predictions out of the total predictions made.

Precision and recall are two metrics which are equally important in evaluating the performance of a model. A model can have higher precision and lower recall or vice versa, this triggers the need for a single metric to measure the performance. The trade-off between precision and recall is found by calculating the F1 score. Precision, recall, F1 score and accuracy have values

between *0* to *1*, with zero being the worst and one being the best possible score for a model. Precision, recall and F1 score are computed separately for each class and are combined by taking the average to find their overall value while accuracy is computed by taking into account all true positives, true negatives, false positives and false negatives. Accuracy is commonly used as a metric for evaluation but when there exist imbalance in the distribution of samples belonging to the classes, which means there exist variations in the number of samples belonging to each class, the results might be misleading. Consider the example of a four class problem with class distribution [*class A: 70 samples, class B: 10 samples, class C: 10 samples, class D: 10 samples*] ,the accuracy of the classifier would be *0.7* by just predicting everything as belonging to class A. But this type of evaluation fails to measure the actual ability of the model to perform a classification task.

In a multi-class classification task, where there exist more than two classes. The overall performance of the system is computed by taking the average of the score associated with individual classes as micro-average, macro-average and weighted average (see Figure 3.27). Micro-average scores are calculated by taking true positives, true negative and false positives of individual classes and combining them to find the overall score for. Macro average is computed by taking the average scores of the individual class metrics. Macro average assigns equal weights to the evaluated metric value for each class, for weighted average we weigh the metric of each class based on the number of entities of each class [86]. Micro-average gives equal importance to each sample prediction, while macro-average gives equal importance to all classes, be it majority or minority. For this study micro average F1-score was chosen as the evaluation metric for the comparison of different models.

Metrics	Formula
Micro-average F1 score	$2\times \left\{ \frac{\left(\frac{tp1+tp2+tp3}{tp1+tp2+tp3+fp1+fp2+fp3}\right)\times\left(\frac{tp1+tp2+tp3}{tp1+tp2+tp3+fn1+fn2+fn3}\right)}{\left(\frac{tp1+tp2+tp3}{tp1+tp2+tp3+fp1+fp2+fp3}\right)+\left(\frac{tp1+tp2+tp3}{tp1+tp2+tp3+fn1+fn2+fn3}\right)} \right\}$
Macro-average F1 score	$\frac{F1\_class1+F1\_class2+F3\_class3}{3}$
Weighted-average F1 score	$\frac{2\times F1\_class1+3\times F1\_class2+4\times F3\_class3}{9}$

Figure 3.27: Formula for variants of averaging class-wise F1 scores for computing the overall F1 score of a model

This study uses a publically available python package known as seqeval [87] for generating the classification report. A classification report is used to measure of the ability of a model to perform classification. It holds information about the class-wise as well as micro, macro and weighted averaged precision, recall and F1 score. Figure 3.28 shows an example of a classification report generated by using seqeval. In the classification report, the IOB2 tags combine individual tokens to form a single entity. If a token with tag "*B-geo*" is followed by another token with tag "*I-geo*", they are considered a single entity in the final classification report. The scores are calculated based on the strict matching evaluation criteria , that is a prediction is considered as true positive only if the entire entity, consisting of multiple chunks are predicted correctly e.g. even though the first word in the entity is classified correctly as "*B-geo*", rest of the two are wrongly classified, the prediction of that entity is considered wrong and therefore the true positives for *geo* is recorded as *0* [88] .

```
y_true = [['B-geo', 'I-geo', 'I-geo', 'B-gpe', 'O', 'O', 'B-org', 'I-org', 'O', 'O', 'B-org', 'O']]
y_p_test =[['B-geo', 'I-geo', 'O', 'O', 'O', 'O', 'O', 'O', 'B-geo', 'B-geo', 'O', 'O']]
```

```
print(classification_report(y_true, y_p_test))
```

```
              precision    recall  f1-score   support

         geo       0.00      0.00      0.00         1
         gpe       0.00      0.00      0.00         1
         org       0.00      0.00      0.00         2

   micro avg       0.00      0.00      0.00         4
   macro avg       0.00      0.00      0.00         4
weighted avg       0.00      0.00      0.00         4
```

Figure 3.28: Strict matching classification report generated by seqeval

3.4.4 *k*-fold cross-validation

Cross-validation is a method of evaluating a model's performance on a smaller data set by the process of resampling. *k*-fold cross-validation is a variation of cross-validation where the entire training data is divided into *k* folds. During training, a single fold is considered as validation data, and the model is trained on the rest of the data. An advantage of this method is that the model gets exposed to variations in the data set. The model's performance might be different for each fold, due to the variations in the distribution of the data. Evaluating the model on different sets of data each time and then taking the average of the model performance helps get a more robust understanding of the model performance.

The steps involved are :

1. Randomly shuffle the whole data set
2. Divide the data set into *k* groups
3. Train the model on (*k-1*) parts of the data and evaluate the model on a single group.
4. The above step is repeated *k* times.
5. Evaluate the model performance by analysing the performance of the model for each fold

There is no formal rule for selecting a value of *k*, but a value between *5* to *10* is usually used. A value of *k* is selected in way that makes each train validation split large enough to generalise the data and a value that reduces the bias and variance in the evaluation scores [89]. Bias refers

to the systematic difference between true value and the predicted value. Variance refers to the difference in scores when the model in trained on a different data set.

3.5 Experiments

This section includes the empirical experiments conducted to report observations to solve our problem and to show if our hypothesis is valid or not.

3.5.1 Experiments done to compare the performance of Bi-LSTM-CRF models when combined with different word vectorisation techniques

The performance of different word vectorisation techniques were compared by feeding the word embeddings generated by each technique (see Sec 3.3.6) to a simple single layer Bi-LSTM-CRF model (see Sec 3.3.7) based on the overall performance as well as class-wise performance in the NER task. The comparative study was conducted on the same manually created data set (see Sec. 2.3). Architecture similar to [17] was chosen as a Bi-LSTM-CRF model has shown to achieve good performance in a majority of NLP task in comparison to a Bi-LSTM softmax classifier as per previous researches [66, 70]. A simple architecture was chosen to make sure that the performance of the model was not impacted by the complexity of the model. This, we believe, would help understanding the variations in the model's performance when we use a different word embedding.

Table 3.5: Hyperparameters used for building the models for comparative study of the word vectorisation techniques

Hyper parameters	Bi-LSTM-CRF model with Keras	Bi-LSTM-CRF model with GloVe embeddings	Bi-LSTM-CRF model with ELMo embeddings
Input dimension	100	100	100
Output dimension	100	100	1024
Number of units in Bi-LSTM layer	50	50	50
Recurrent dropout	0.1	0.1	0.1
Number of units in time distributed dense layer	50	50	50
Optimizer	RMSprop	RMSprop	RMSprop
Loss	crf.loss_function	crf.loss_function	crf.loss_function

The model consist of an input layer, an embedding layer, two hidden layers, a Bi-LSTM layer and a time distributed dense layer and finally a CRF output layer as described in (see Sec 3.3.7). The hyperparameters used for building the models are encapsulated in Table 3.5. The data given as input is pre-processed following the steps described in Sec 3.2. We specify the input dimension equal to the unique number of words in the training data + two and input length = *100*. A single layer bidirectional LSTM with *50* neurons was used, as per previous researches [90] [17] it was found that the model performance is not dependent on the number of units in the hidden layer hence we have initialised the number of units in Bi-LSTM layer randomly to *50* to keep the architecture simple. Since aim of this study to compare the effect of different word vectorisation methods on a simple and common Bi-LSTM-CRF model the parameters where randomly chosen and fixed for the experiments and hyperparameter tuning was not performed.

Output embedding dimension of *100* was chosen for embeddings generated by the embedding layer and the GloVe embeddings, while ELMo language model produces embeddings of dimension *1024* by default and this was left unchanged. The embedding dimension were chosen as *100* for keras embedding to match with the embedding dimension of GloVe embeddings, since ELMo embeddings have *1024* by default and is not available in *100*, it was used. The data given as input is pre-processed following the steps described in Sec 3.2. We then trained the models in mini-batches of size *32* for *10* to *50* epochs with a step size of *10* between them. The CRF Loss function was used to measure how efficiently the model performs, and this was optimised using the "RMSprop" optimiser (see Sec 3.3.5, 3.3.4).

Since there exist randomness in the results obtained during each run, we used the random seed to fix this. By investigating further it was found that the randomness was because of the stochastic nature of deep learning, during the training process, where each node is initialised with random weights as well as due to the randomness in optimisation. Randomness also occurs due to the random nature of the drop-out layer. The dropout layer drops out a specified percentage of nodes during each run and the nodes getting dropped keep on varying. Additionally Keras gets its source of randomness from the NumPy random number generator while TensorFlow gets the randomness from its own random number generator. Connecting to the runtime in Google Collab can also be a contributing factor to the randomness. To handle the randomness, we set the random seed for python, NumPy random number generator, TensorFlow random number generator, set the seed of Keras initialiser. Additionally, we forced

TensorFlow to use a single thread for execution, as multiple threads are also sources of non-reproducible results. It could also be due to the third party library used for building the CRF layer. Hence, we averaged the model results for each epoch for five runs to get more reliable results similar to the approaches taken by previous research in the field [18, 70, 72, 91]. The mean and standard deviation of the distribution of F1 score for five runs across the epochs is reported. Also, we used *k* fold cross-validation with four folds for evaluating the model performance on each run to introduce the model to variation in the data set. It should be noted that in classification problem with unbalanced data set stratified *k*-fold is usually used. In stratified *k* fold during the splitting of training and test data, the data set gets split in a way that the distribution of all tags are uniform in the test and training data. But in sequential data where each sample is composed of a sequence of words and sequence of targets the split happens at sentence level and not at the word level, e.g. *sentence1, sentence5, sentence6* becomes training data while *sentence2, sentence3* becomes test data. But the imbalance is in terms of number of elements belonging to each tag. Intuitively , since this is a sequence modelling task with a set of sequences of words stratified *k*-fold was not used. If it was a single word classification task without the sequence it would have been possible to do a stratified *k*-fold.

We then report the performance of Bi-LSTM-CRF with the three different word embeddings by training each model for *10, 20, 30, 40, 50* epochs. The class-wise prediction performance using each word vectorisation technique is also reported. The epoch number at which the models gave high micro-F1 score was chosen. The model was trained on the entire training data at this setting and was evaluated on the test data. The model's performance on test data was reported along with the results obtained by other Bi-LSTM-CRF NER models as per previous researches. Owing to the stochastic nature of deep learning models, the evaluation on test data was also repeated five times and the average and standard deviation was reported without making any other changes in the model definition or the data set.

3.5.2 Experiments done to build an efficient automatic Named Entity Recogniser

The best performing model was then selected based on their performance with the validation data. Also, for building an efficient NER system we tuned two hyper parameters of the best performing model on a small search space (see Table 3.6) due to the time constraints. Optimiser waa chosen as a hyperparameters based on the study performed by Reimers et al. [90] on NER performance on CoNLL 2003 data set [20], where the impact of optimisers on the performance

of NER model using Bi-LSTM-CRF was found to be high. Recurrent dropout was chosen as a hyper parameter based on previous researches that suggested that [92, 93], recurrent dropout resulted in improved model performance.

Table 3.6: Hyperparameter search space used for tuning the best model found from the comparative study

Hyperparameter	**Search space**	**Final value selected**
Optimiser	adam [94], Nadam [95], RMSprop	adam
Recurrent dropout	0.1, 0.2, 0.3	0.3

Training was performed for each combination of hyperparameter following the same steps as the previous experiments keeping all other hyperparameters same as Table 3.5. The hyperparameter combination for which the model gave highest performance score on the training data, was fixed and was evaluated on the test data and the scores were reported.

Chapter 4 Results and discussion

This chapter reports the results obtained by conducting the experiments explained in the methods section. These results are also discussed and analysed further in this section. We also compare the results obtained through this study with previous researches in the field and analyse how they deviate from those expectations. The theory behind the models used as well as word vectorisation techniques used are discussed in Chapter 3.

4.1 Performance comparison of NER with different word vectorization techniques

In order to expose the models to variations in data, the models were trained using *4*-fold cross-validation, and the scores were calculated by taking an average of micro-F1 scores for each fold. Micro-average F1 score was used as the metric for evaluating and comparing the overall model performances as it gives equal importance to each sample prediction. Table 4.1 reports the overall micro average F1 score of performing NER with the standard deviation corresponding to the five repetitions of the experiments for the three models for epochs ranging from *10* to *50*. It could be observed that the Bi-LSTM-CRF model with *100-dimensional* GloVe embeddings gave the best micro-F1 score of 0.869 when trained for *50* epochs. Hence this model trained for *50* epochs was selected for further optimization and creation of an efficient NER system in Sec. 4.2. The best scores for each model composed of Bi-LSTM-CRF and corresponding word vectorization technique when trained for *10* to *50* epochs have been highlighted in bold in Table 4.1 for reference.

Table 4.1: Overall micro F1 scores with the standard deviation of three models for epochs ranging from 10 to 50 evaluated using 4-fold cross-validation repeated five times. The values in bold correspond to the best performance score of each model trained for that many epochs

Model	Epochs	Micro-F1 score
Bi-LSTM-CRF (Keras embedding layer)	10	0.774±0.0076
	20	0.845±0.0023
	30	0.841±0.0026

	40	**0.846±0.0009**
	50	0.834±0.0034
Bi-LSTM-CRF (GloVe embedding)	10	0.808±0.0027
	20	0.861±0.0007
	30	0.856±0.0012
	40	0.847±0.0026
	50	**0.869±0.0010**
Bi-LSTM-CRF (ELMo embedding)	**10**	**0.846±0.0093**
	20	0.844±.0056
	30	0.842±0.0088
	40	0.831±0.0064
	50	0.836±0.0067

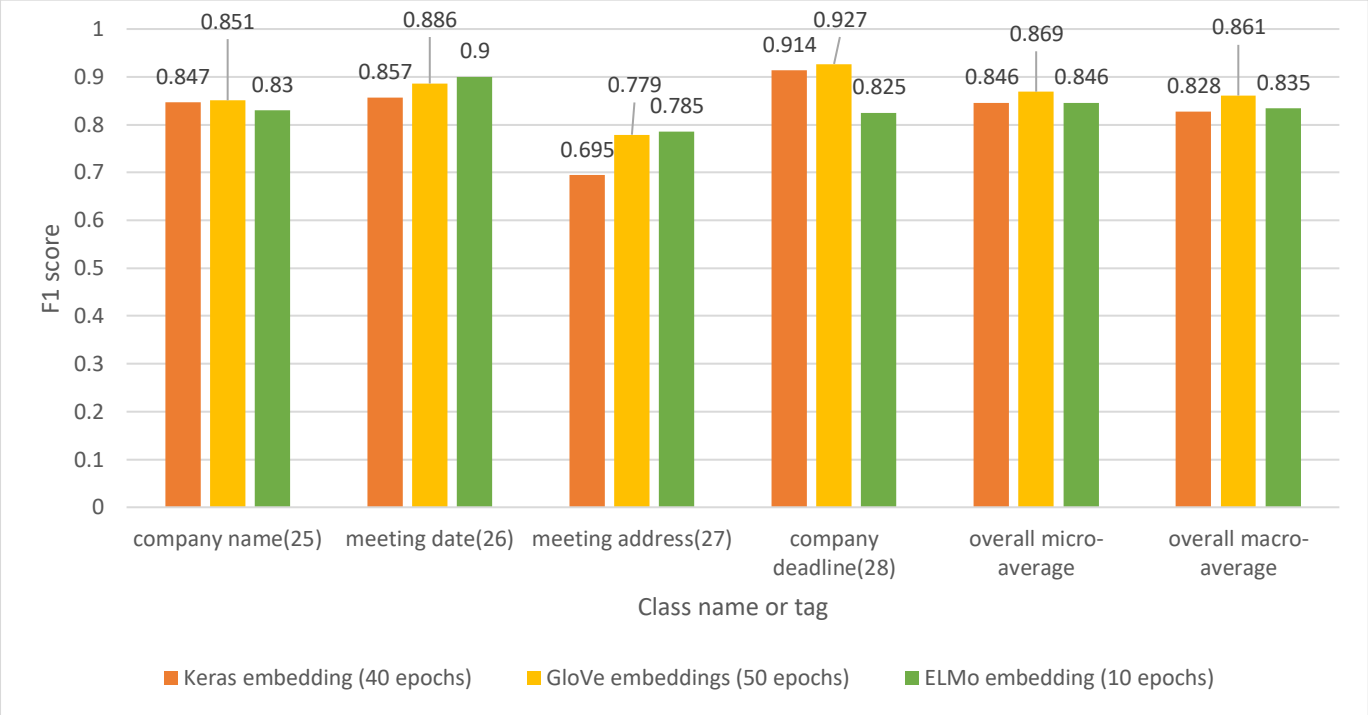


Figure 4.1: Comparison of class-wise and overall micro and macro average F1 score of best performing models on training data with the corresponding epoch number

The class-wise performance of the models was also recorded along with the overall performance score for epochs ranging from *10* to *50* to understand the effect of different word vectorization techniques used for NER. Figure 4.1 shows the results of class-wise and overall F1 scores of the best models in performing NER with the training data. Micro-averaging was initially chosen as the metric for averaging the individual F1 scores of the classes and computing the overall score. This was done to avoid misleading performance scores evaluation

by taking the simple average of performance scores of each class, also known as macro-averaging. This could be inaccurate in situations where the majority class have a lower performance in comparison with the minority class. Macro-averaging gives equal importance to each class, be it majority or minority, which would then lead to a higher overall F1 score, but in reality, the model is performing bad with a majority of samples. Analysing the class-wise F1 score shows that the majority classes, company name (25), meeting date (26), and company deadline (28), have higher F1 score in comparison with the minority class, meeting address (27) (see Figure 4.2), hence preventing the chance of inaccurate performance score while using macro-averaging. Since both these scores seem to be interesting to analyse. Further in the analysis of results, we will be using both the micro-average and macro-average scores as the evaluation metrics as suggested by Yang et al. [96].

Bi-LSTM-CRF model with GloVe embeddings gave the highest overall micro-F1 score of 0.869 and an overall macro-F1 score of 0.861 out of the three models used for the comparative study. This result is a deviation from the expectation that using deep contextualised word embeddings like ELMo would lead to better model performance. But it is interesting to note that Bi-LSTM-CRF with ELMo needed fewer epochs to converge and achieve good results of overall micro F1 score of 0.846 and macro F1 score of 0.835 in comparison to the other models which took *50* epochs for Bi-LSTM-CRF with GloVe embeddings to achieve an overall micro F1 score 0.869 and overall macro F1 score of 0.861 and *40* epochs for Bi-LSTM-CRF with Keras embedding layer to achieve an overall micro-F1 score of 0.846 and macro F1 score of 0.828. The results demonstrate that Bi-LSTM-CRF models with GloVe embeddings also gave better or almost equal class-wise F1 scores in comparison to Bi-LSTM-CRF models performing NER using Keras embedding and ELMo embeddings for word vectorisation for three out of four classes with 0.851 versus 0.847, 0.83 for company name (25), 0.927 versus 0.914, 0.825 for company deadline (28) 0.779 versus 0.695, 0.785 for meeting address (26). Additionally, it could be observed that the Bi-LSTM-CRF model with ELMo embeddings gave the next best class-wise F1 score for meeting date (26) with an F1 score of 0.9 and meeting address (28) with an F1 score of 0.785. The lowered performance in classifying samples belonging to the classes company name (25), company deadline (28) in comparison to the other models might be the reason for an overall low score for the Bi-LSTM-CRF model with ELMo embeddings. ELMo being a contextualised word embedding, the existence of two different labels meeting date (26) and company deadline (28), which are in the same format but different labels, might be a reason for the confusion in the context based models resulting in a low F1

score of 0.825 for company deadline (28) in comparison to meeting date (26) with an F1 score of 0.90. Analysing the overall macro and micro F1 score of Bi-LSTM-CRF with ELMo embeddings shows that the ELMo model has a better performance score than Bi-LSTM-CRF with Keras embedding layer in terms of macro-average F1 score with scores of 0.835 for ELMo and 0.828 for Bi-LSTM-CRF Keras embedding layer, while they have equal micro F1 scores of 0.846.

The lowered performance of the model using ELMo embeddings may also be due to the high dimensional word embeddings, with a dimension of *1024*, which is the default output embedding dimension of using the pre-trained ELMo model. This high dimensionality might give rise to the problem of "Curse of Dimensionality" [35, 97]. An increase in the dimensionality of the model leads to more features, thus triggering the need for more data to capture all possible combinations of features. In our case, though output dimensionality is increased, the number of training samples remains the same, which might be the reason for low F1 scores for NER using Bi-LSTM-CRF and ELMo embeddings. The lower dimensionality of *100* of the output embedding could also be a reason for the improved performance of Bi-LSTM-CRF with GloVe embeddings. In addition, through analysis of the data set, it was found that out of *1413* unique words in the training data, *619* words had embeddings in the GloVe embedding matrix. In order to analyse the curse of dimensionality, we perform a secondary experiment with GloVe embeddings of smaller dimensions with the same model and same data set to observe the effect on results.

The whole data set was divided into training and test data in a *90:10* ratio as part of pre-processing. The initial data set had *747* sentences, with each sentence having *100* words. After the test train split, the training data was composed of *672* sentences and the test data was composed of 65 sentences. The resulting training data set had *1413* unique words, and the test data set had *707* unique words. In order to understand the class-wise performance better, it is important to include the tag distribution in the training and test data set. This is included in Figure 4.2 and Figure 4.3. It is also important to note that the performance on the training data was evaluated by using *4*-fold cross-validation, and this was repeated five times. So the distribution of classes in each fold might be different from the distribution of classes in the whole training data.

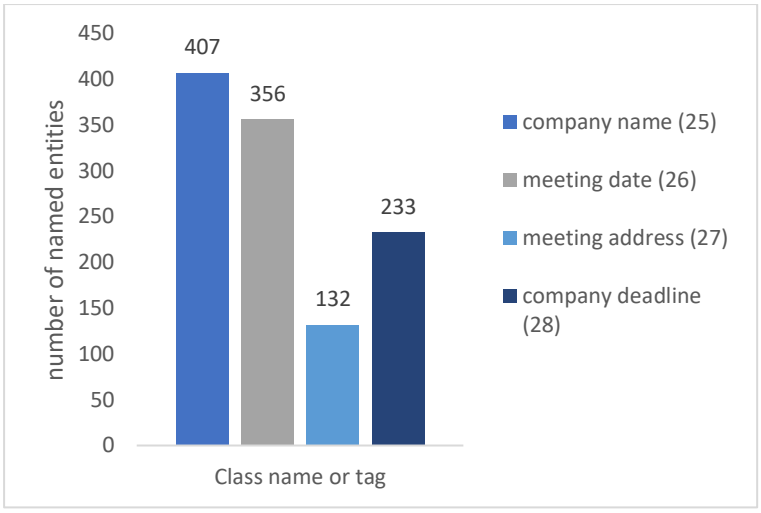


Figure 4.2: Distribution of tags in the training data without including the "O" tag

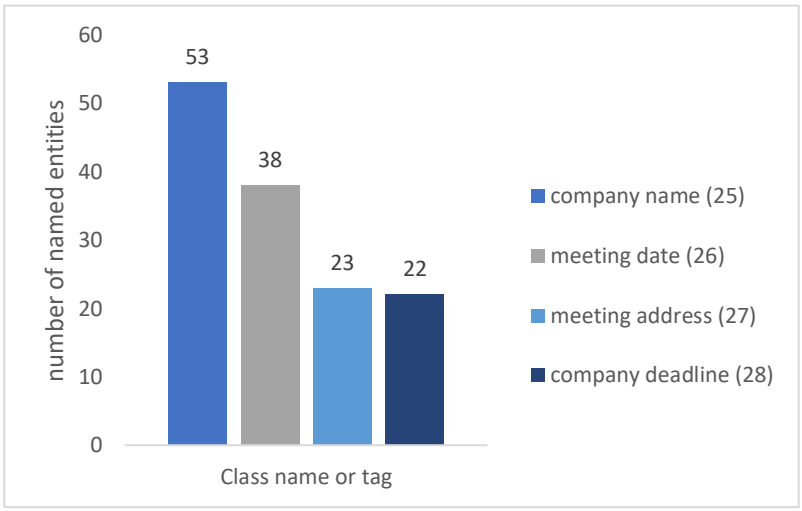


Figure 4.3:Distribution of tags in the test data without including the "O" tag

Figure 4.1 shows that best models in the three cases which gave the highest scores for the classes company deadline (28) and meeting date (26) and least performance in predicting entities belonging to the class meeting address (27). The low performance in predicting entities belonging to the tag meeting address (27) might be because of the low number of samples in the training data belonging to that class or tag. Intuitively it proves the point that since there exist a lower number of samples belonging to that category which can be called a minority, the model is not well exposed to the category and hence performs badly in predicting samples belonging to it. On the other hand, the tag categories for which the models performed the best, company deadline (28) and meeting date (26), deviates from the theory used in explaining the reason for the bad performance in predicting the tag meeting address (27). If that was the case, the model should have shown the best performance in predicting entities belonging to the tag

company name (25), followed by meeting date (26) and company deadline (28). Another hypothesis which we could infer from the good results could be the fact that the model has good performance in predicting categories that have entries in an alphanumeric pattern like "*13 November 2021*". Both company deadline (28) and meeting date (26) have entities in this format. Overall it could be observed that the best model Bi-LSTM-CRF with GloVe embeddings training for *50* epochs managed to give the best F1-score for the NER task as well as in predicting the classes except for the minority class without performing any hyperparameter tuning and using a simple base model. It managed to improve the performance of using a simple Keras embedding layer with an overall micro F1 score of 0.846 by 0.023 to 0.869, and the overall macro F1 score was improved from 0.828 to 0.861. This also outperformed the best performing Bi-LSTM-CRF NER using ELMo embeddings by 0.02 in terms of overall micro F1 score and 0.03 for macro F1 score on the training data (see Figure 4.1).

Table 4.2: Overall micro and macro F1-score with standard deviation for the best performing models on the test data for experiments repeated five times

Model	Epochs	Overall micro F1-score	Overall macro-F1 score
Bi-LSTM-CRF (Keras embedding layer)	40	0.858±0.004	0.848±0.004
Bi-LSTM-CRF (GloVe embedding)	50	0.868±0.009	0.872±0.013
Bi-LSTM-CRF (ELMo embedding)	10	0.796±0.008	0.776±0.008

Table 4.2 presents the performance of the best models found by cross-validation on the training data (see Table 4.1) when applied to the unseen test data. It could be observed that Bi-LSTM-CRF with GloVe embeddings, when trained for *50* epochs which gave the best overall micro F1 score of 0.869 and a macro-F1 score of 0.861 on the validation data, managed to achieve an overall micro F1 score of 0.868 and macro F1 score of 0.872 on the test data (Table 4.2).

It could be observed that the test score in terms of both micro and macro average is slightly higher than validation score in Bi-LSTM-CRF with Keras embedding layer with micro F1 score of 0.848 and macro F1 score 0.858 in comparison with overall micro F1 score of 0.846 and macro F1 score of 0.828 on the validation data. This could be because of the test train split chosen for the study, where *90%* of the data was taken as the training data, and *10%* of the data was chosen as the test data. This *10%* of data chosen as the test data might be favourable to the

model and easy to predict, leading to better results in comparison to the large number of training data that has more variance involved. Repeating the experiments with another ratio of test train split could be interesting to explore in the future.

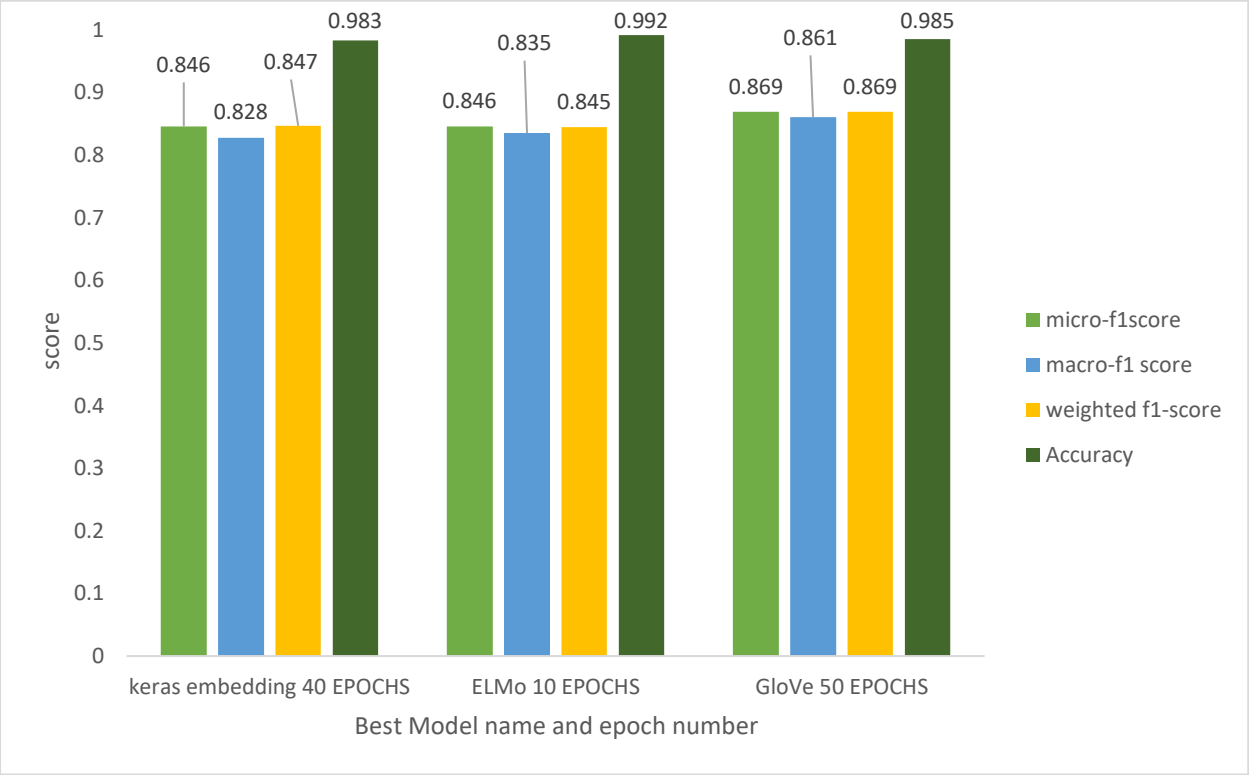


Figure 4.4: Comparison of evaluation metrics based on the scores for the best performing models for that many epochs on the training data evaluated by four-fold cross-validation and experiments repeated five times

Figure 4.4 shows the performance scores of the best performing models when trained for epochs ranging from *10* to *50* on the training data. The scores are reported in two main metrics accuracy and F1 score. The models gave the highest score for the metric accuracy. Accuracy gives highest importance to true positives and true negatives in comparison to false negatives and false positives. So, the inability of a system to classify members correctly to a class is not given importance. F1 score, on the other hand, gives more importance to false negatives and false positives. Hence F1 score was considered as a more accurate measure. There exist three variants of averaging F1 scores of individual classes, and they show slight variations (see Sec. 3.4.3). The weighted average F1-scores reported a higher value in most cases, but this might also be misleading as this overall F1 score calculation biased to the number of samples belonging to a class. The higher value for this metric might be because of a better F1 score for the samples belonging to the majority class, the class to which a higher number of samples belong and hence were weighted more than the others leading to a high overall weighted F1

score. On the other hand, the macro-F1 score gave the lowest score. This metric is strict and gives equal importance to all classes, as it computes the overall F1 score by simply taking the average of F1 scores of individual classes, thus giving equal importance to every class, be it a minority or majority. Micro-averaging, on the other hand, gives equal importance to the prediction of each sample, and therefore false negatives are also taken into account. As discussed earlier, both micro-averaging and macro-averaging of F1 scores seems interesting and will be used simultaneously.

In the above section, we compared the three models and discussed the differences in results. In this section, we will discuss the results and observation of individual models and try to understand their performances which will help in understanding their differences and advantages, which can be interesting for future work.

Word vectorization by Keras embedding layer

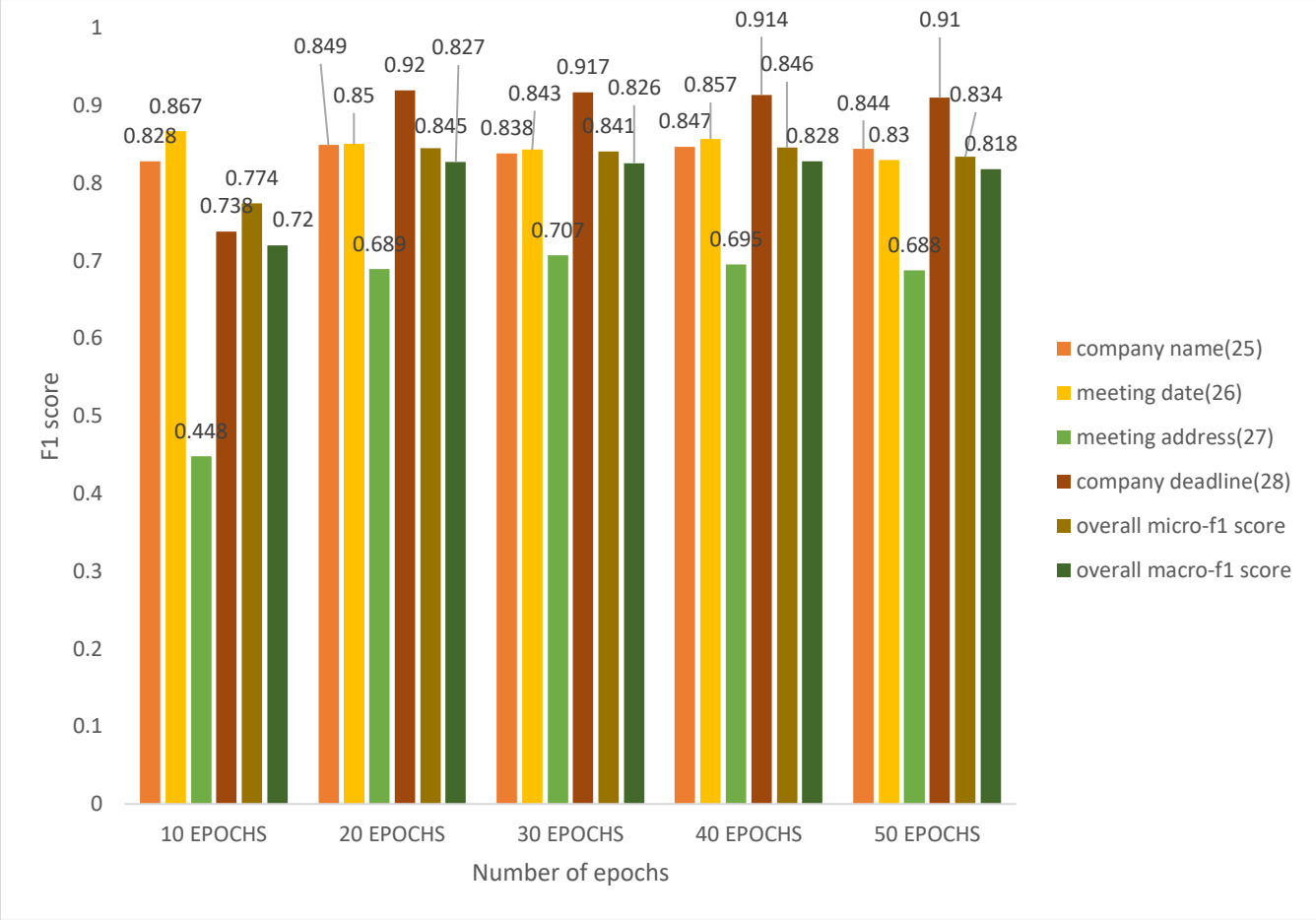


Figure 4.5: Class-wise and overall micro and macro F1 score for Bi-LSTM-CRF with Keras embedding layer on training data for epochs ranging from 10 to 50.

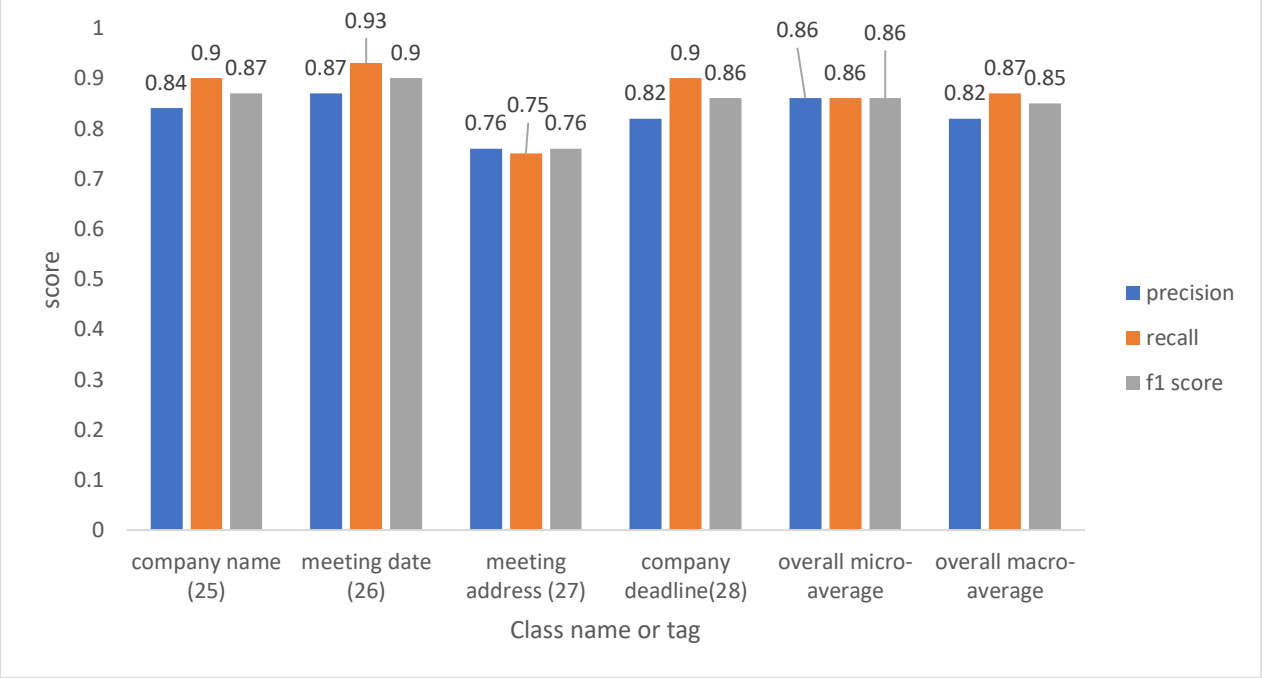


Figure 4.6: Class-wise and overall micro and macro average scores of performance on test data for the best performing model using Keras embedding layer (40 epochs)

Figure 4.5 shows the class-wise and overall F1 score of performing NER using Bi-LSTM-CRF and Keras embedding layer when trained for epochs ranging from *10* to *50*. Overall, the model gave the highest micro-F1 score of 0.846±0.0009 and macro F1 score of 0.828±0.0009 the training data when trained for *40* epochs, using *4*-fold cross-validation, repeated five times. The classification score for the classes meeting date (26) and company deadline (28) were the highest with F1 scores 0.9 and 0.86 in comparison to other classes, while the class meeting address (27) seems to be the most challenging one to classify with an F1-score of 0.76.

Figure 4.6 presents the overall and class-wise score for training the model for *40* epochs on training data and evaluating on unseen test data. Overall the model gave a micro-F1 score of 0.86±0.004 and macro F1 score of 0.85±0.004. In terms of evaluating the model performance based on its ability to classify samples belonging to a class, the model reported the highest F1 score in classifying samples or named entities belonging to the class meeting date (26). It has a recall value of 0.93, meaning that the model correctly classified 93% of the named entities belonging to the class meeting date (26). On the other hand, the model gave the worst F1 score of 0.76 for the class meeting address (27), which is similar to the model's performance with the training data. The reason for difficulties in classifying entities belonging to the class meeting address (26) might be the low number of samples in training data belonging to that particular class (see Figure 4.2).

Word vectorisation with 100-dimensional GloVe embeddings

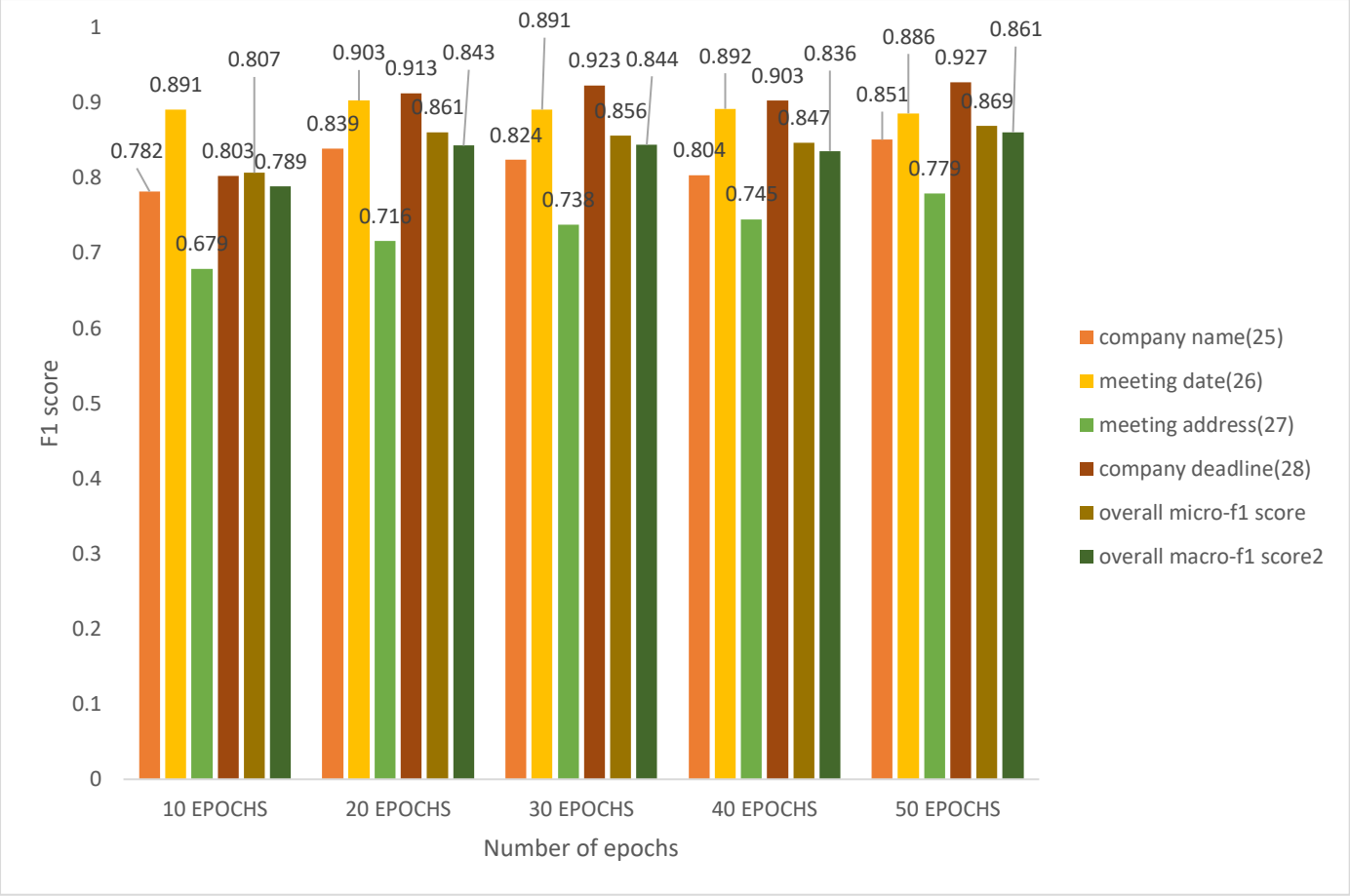


Figure 4.7: Class-wise and overall F1 score for Bi-LSTM-CRF (100-dimensional GloVe embeddings) for epochs ranging from 10 to 50.

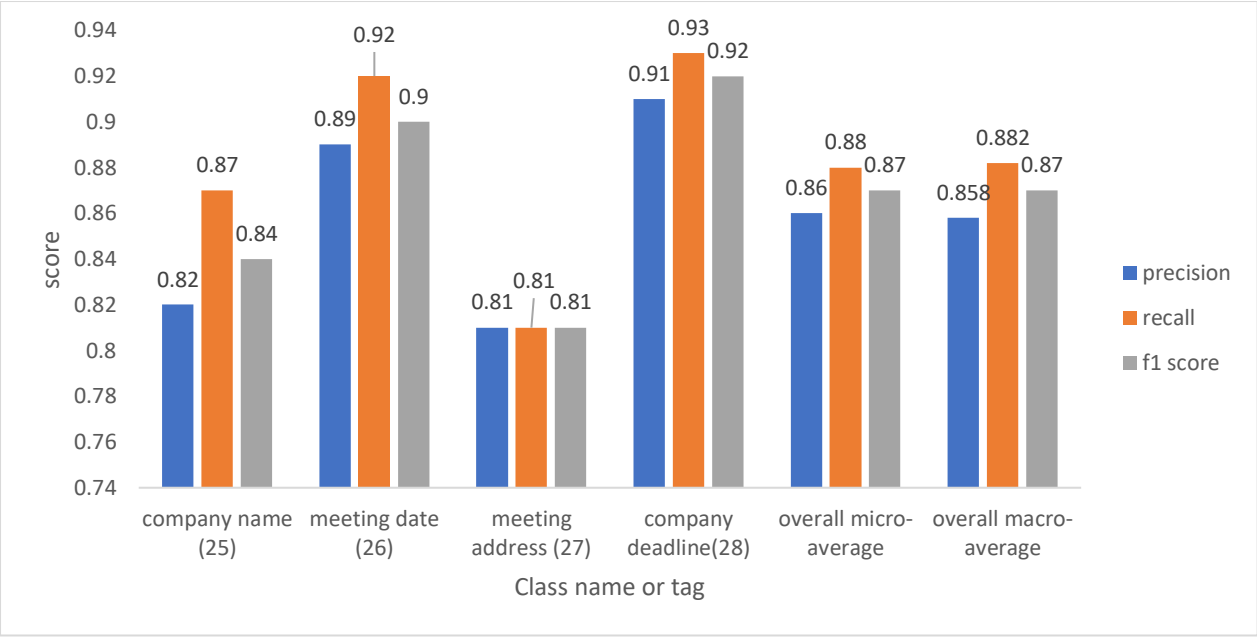


Figure 4.8: Class-wise and overall score of performance on test data for the best performing model using 100-dimensional GloVe embeddings (50 epochs)

Figure 4.7 illustrates the class-wise and overall performance of the Bi-LSTM-CRF NER model with *100-dimensional* GloVe embeddings trained for epochs ranging from *10* to *50* in terms of F1 score. The model managed to achieve the best overall micro F1 score of 0.869±0.0010 and overall macro F1 score of 0.861±0.001 (see Figure 4.7) on training data when trained using *4*-fold cross-validation repeated for five times for *50* epochs. It could also be observed that the best model achieved the best class-wise F1 score in classifying named entities belonging to the tag company deadline (28). It is interesting to note that not just the overall performance but the model also shows a good class-wise F1 score when trained for *50* epochs. As observed earlier, the model finds it challenging to classify named entities belonging to the class meeting address (27), which accounts for the lowest number of samples in the training data.

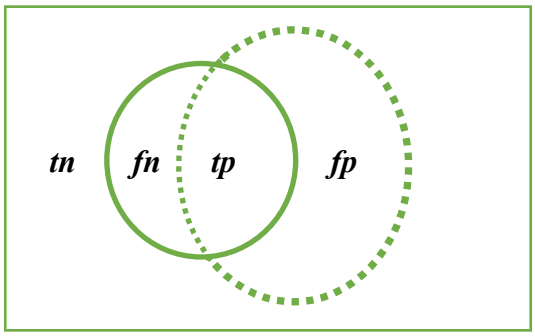


Figure 4.9: Figure showing high recall and low precision, where tn: true negatives, fn: false negatives, tp: true positives, fp: false positives. The solid circle represents the ground truth and th edotted line cirrcle represents the prediction

Figure 4.8 shows the class-wise and overall performance of the best performing model on the test data in terms of precision, recall and F1 score. The best performing model was selected based on the performance score on the validation data set. The model, when trained for *50* epochs and evaluated on the test data, gave an overall micro and macro F1 score of 0.87 with an overall micro-average and macro-average precision and recall of 0.86, 0.88. The metrics show that the model managed to achieve a high micro and macro recall in comparison to the micro and macro precision, meaning the model managed to achieve a large number correct predictions or true positives out of the total ground truth, which includes the true positives and false negatives (see Figure 4.9). F1 score is used as a measure to find the balance between recall and precision as both these metrics are important for the good performance of a model. Higher the F1 score for a class, better the performance of the model for that class. In terms of class-wise performance on the test data, the model managed to give the highest F1 score in

classifying named entities belonging to the class company deadline (28). This behaviour is similar to what was observed in the training data suggesting there is a good generalisation.

Word vectorization by ELMo embeddings

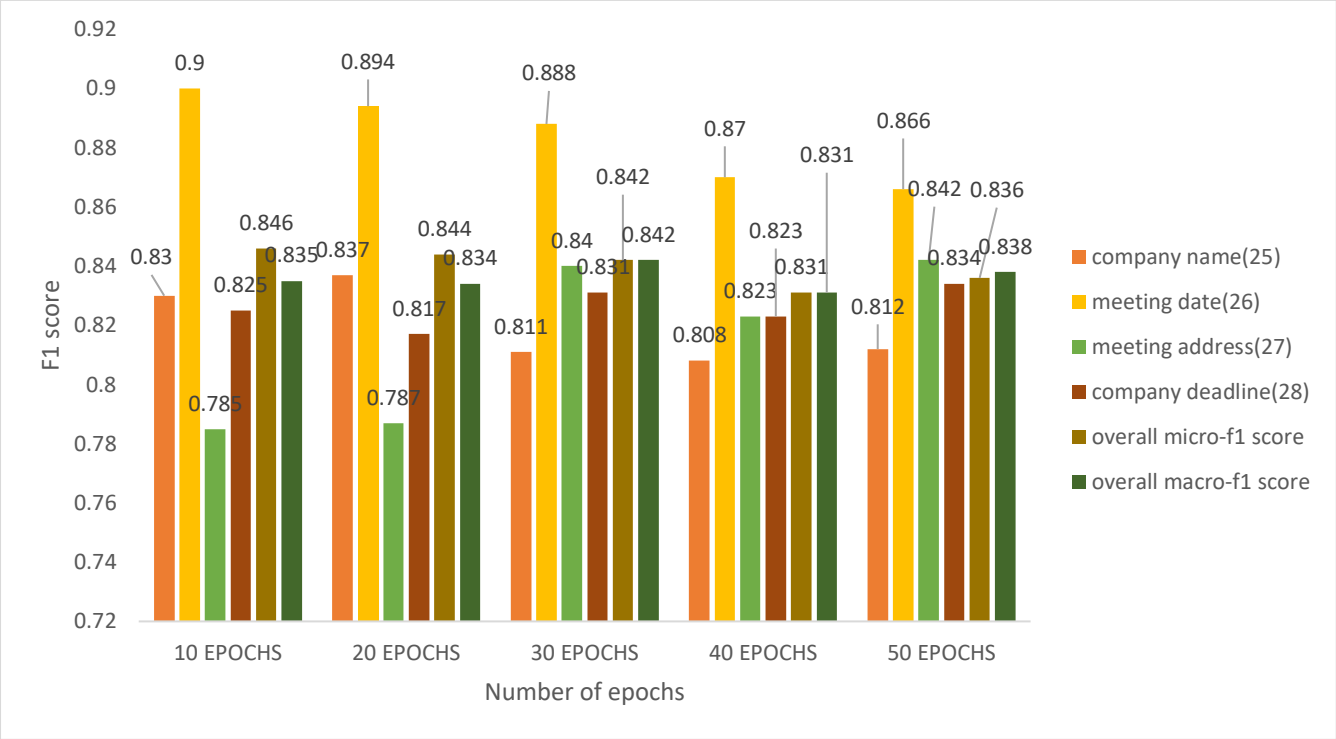


Figure 4.10: Class-wise and overall F1 score for Bi-LSTM-CRF (ELMo embeddings) for epochs ranging from 10 to 50

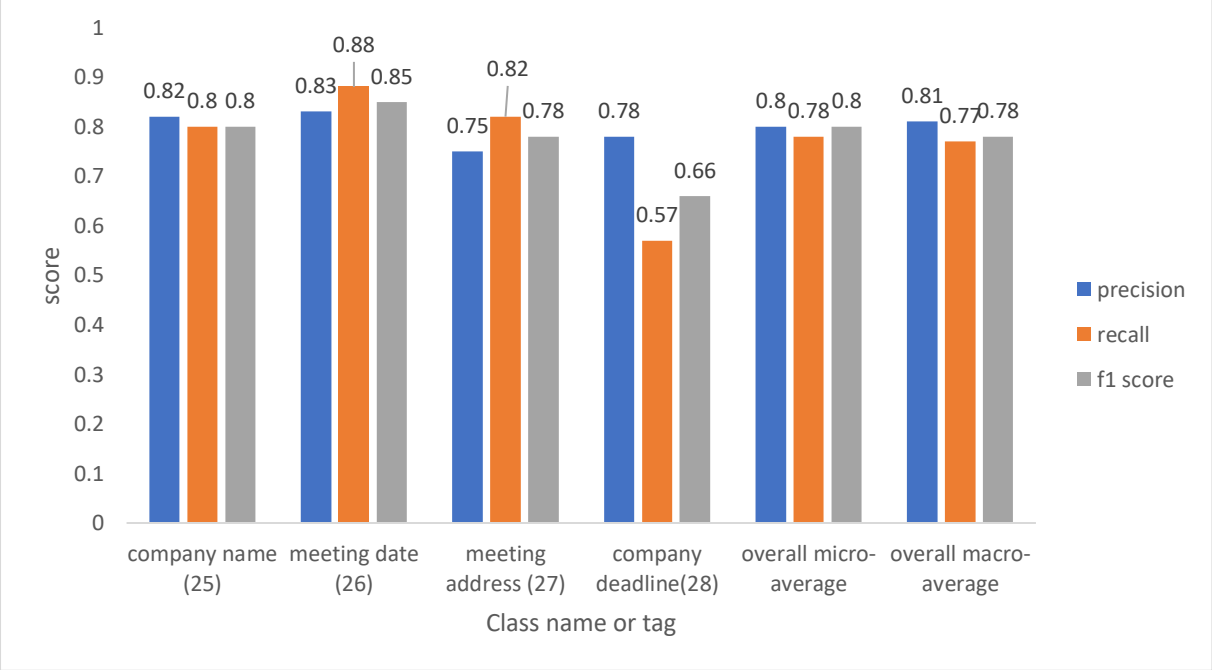


Figure 4.11: Class-wise and overall performance score in macro and micro average on test data for the best performing model using ELMo embeddings (10 epochs)

Figure 4.10 and Figure 4.11 shows the class-wise and overall performance scores of NER using Bi-LSTM-CRF with ELMo embeddings on training and test data in terms of precision, recall and F1-score. The overall F1 score used is the micro-F1 score for the ELMo model. The high performance score of NER using ELMo embeddings for minority classes in most cases (*30*, *40* and *50* epochs) in comparison to the majority classes on the training data triggers to take into account the micro F1 score for evaluating the overall model performance and finding the number of epochs at which the model gave the best performance. Figure 4.10 shows the results on the training data evaluated by *4*-fold cross-validation and taking the average of repeating the *4*-fold cross-validation five times. It could be observed that the model gave the best overall micro F1-score of 0.846 when trained for ten epochs with a standard deviation of 0.0093 for the five repeats (see Table 4.2). The model gave the best class-wise F1-score for the class meeting date (26), with an F1-score of 0.85 and the class meeting address (27) was the most challenging to classify with the lowest F1-score of 0.79.

On the test data, the Bi-LSTM-CRF NER model using ELMo embeddings gave an overall micro F1 score of 0.796 when trained for ten epochs with a standard deviation of 0.008 and an overall macro F1 score of 0.776 with a standard deviation of 0.008 by repeating the testing five times (see Table 4.2), and the model managed to classify named entities belonging to the class meeting date (26) with the highest F1 score of 0.85. It is interesting to note that though both meeting date (26) and company deadline (28) has a similar format, "*13 November 2021*", the model managed the classify named entities belonging to the class meeting date (26) with high F1 score while those belonging to the class company deadline (28) was classified with a low F1 score of 0.66. This might be due to the confusion created in contextual embeddings due to the existence of similar embeddings with two different labels, which is similar to the behaviour seen on the training data.

4.2 Results of experiments done to build an efficient automatic Named Entity Recogniser

The Bi-LSTM-CRF model with *100-dimensional* GloVe embeddings trained for *50* epochs, which was found to be the best model, was further tuned using the hyperparameter space in Table 3.6. The figures Figure 4.12, Figure 4.13 shows the comparison of the performance of the best model with the tuned model on the training data and the test data with the untuned model with *100-dimensional* Glove embeddings. Figure 4.12 shows that hyperparameter

tuning did not result in a huge improvement in model performance score with an overall micro-average F1 score of 0.875 vs 0.868 and macro-average F1 score of 0.86 vs 0.861 was obtained when the model was trained for *50* epochs using adam optimiser and a recurrent dropout of *0.3* on training data. A micro-average F1 score of 0.87 vs 0.87 and a macro-average F1 score of 0.868 vs 0.87 was observed on the test data. Lack of improvement in performance may be due to the smaller search space used for hyperparameter tuning.

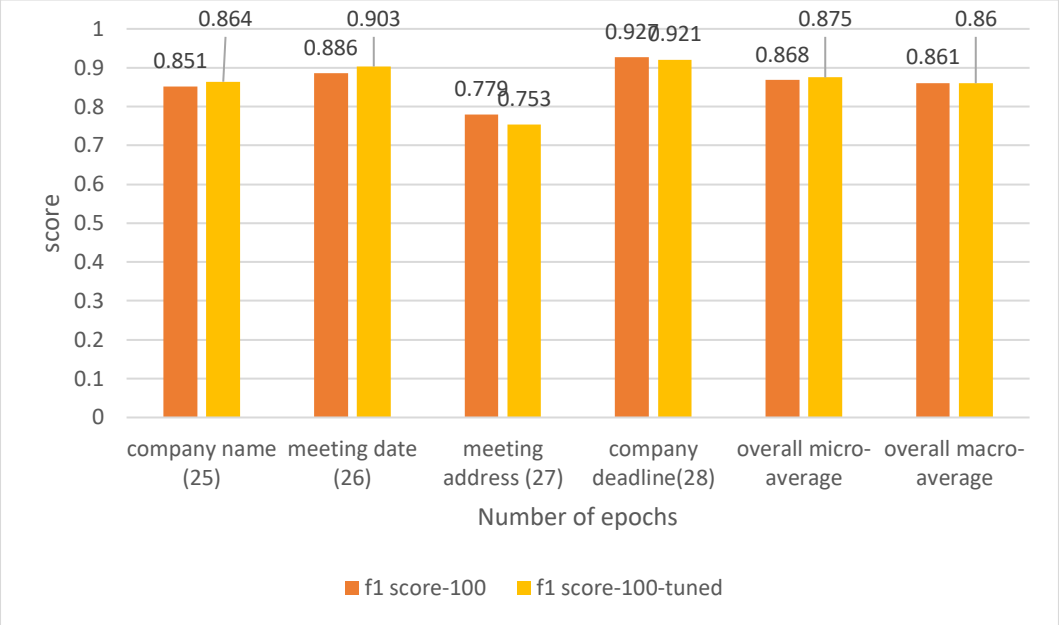


Figure 4.12: Comparison of overall and class-wise performance scores of the Bi-LSTM-CRF model with 100-dimensional GloVe embeddings and the tuned model on the training data

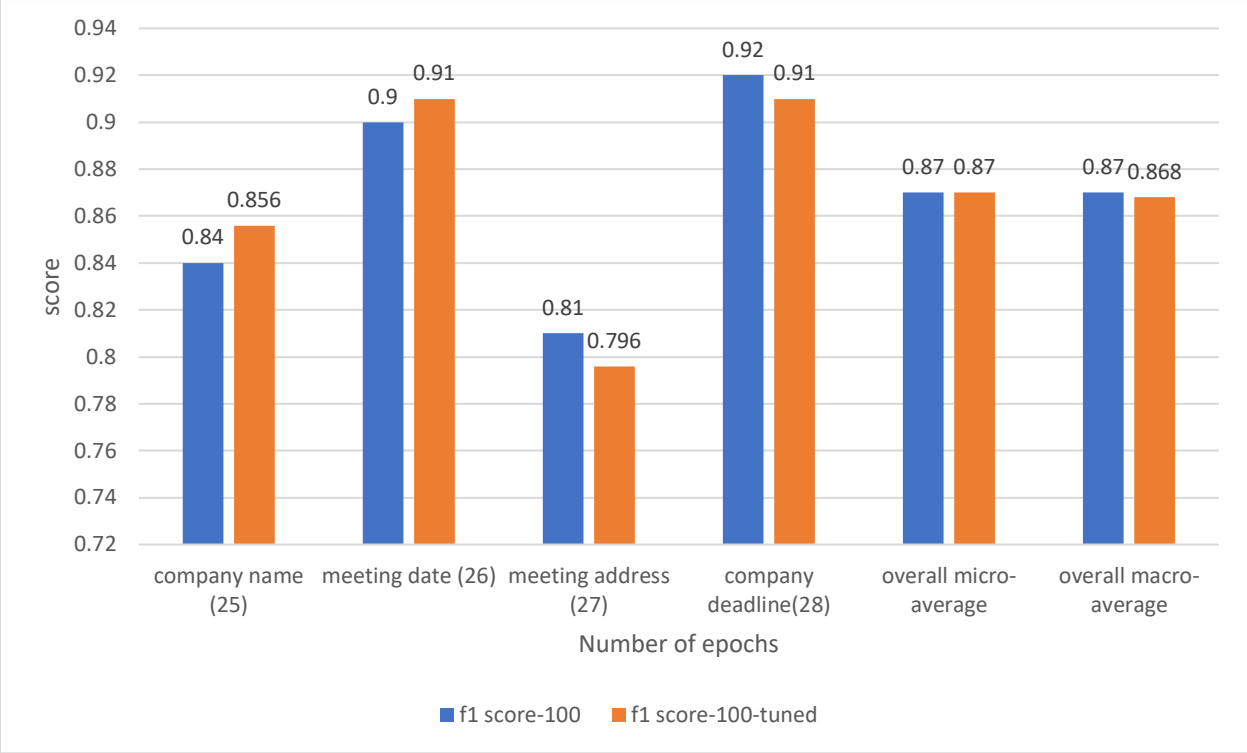


Figure 4.13: Comparison of overall and class-wise performance scores of the Bi-LSTM-CRF model with 100-dimensional GloVe embeddings and the tuned model on the test data

Summary of results

Evaluating the performance of the best models found on the unseen test data, we can see that for models with word vectorisation by Keras embedding layer, and GloVe embeddings gave high recall, meaning that the models managed to make more correct predictions, in the form of true positives out of the total ground truth. In order to find the balance between recall and precision for evaluating the models, we use the F1 score as the measure. Comparing the overall F1-scores of the models on the unseen test data also shows that the models managed to achieve the best F1-score with a Bi-LSTM-CRF model with *100*-dimensional GloVe embeddings, as shown in Table 4.2, this is in line with the results on the 2020 i2b2/VA data set [23, 98]. It should be noted that during the process of training and evaluating the models, we faced challenges in reproducing the results of experiments due to the stochastic nature of deep learning models. The artificial neural network operates on randomness, from random initialization of weights to random shuffling of the data while training [99]. This leads to variation in results and performance of the model when the model is trained and evaluated each time on the same data. Also, tuning the best model on a small hyperparameter search space did not result in improvement in model performance.

Secondary Experiment

As proposed earlier the curse of dimensionality due to the high dimensionality of the output vector produced by ELMo embedding may be the reason for the result contradicting the hypothesis that contextual word embeddings would result in the best model. Hence it looks interesting to analyse the effects of using embeddings of different dimensions on the same data set and base model and observe the variations in results. This was thus performed by comparing the model performance of using GloVe embeddings in *50* and *100* dimensions. It was observed that training a Bi-LSTM-CRF NER with *50*-dimensional GloVe embeddings resulted in an overall average micro-F1 score of 0.8653±0.0013 on the training data using *4*-fold cross-validation repeated five times and 0.876±0.004 on test data for five repeats. It was also observed to achieve an overall macro F1 score of 0.85 on the training data and 0.88 on the test data.

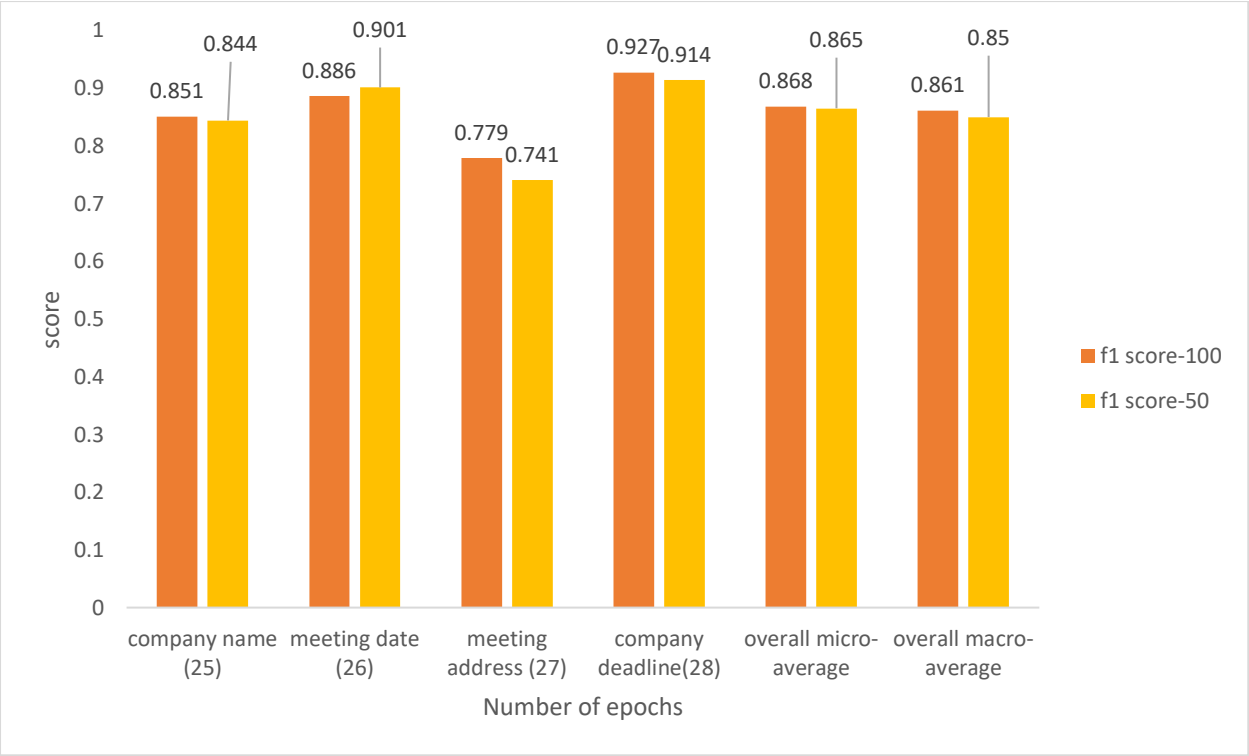


Figure 4.14: Class-wise and overall F1 scores of Bi-LSTM-CRF model using GloVe embeddings of 50 dimensions and 100 dimensions on training data

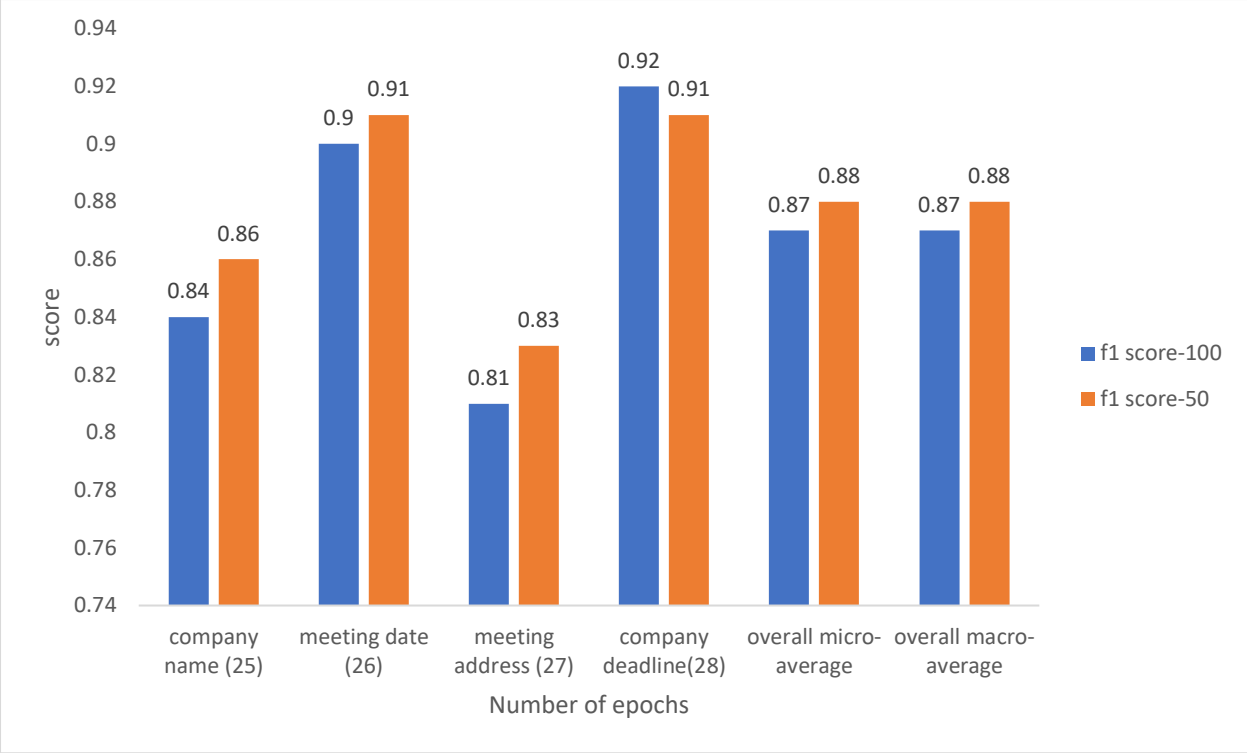


Figure 4.15:Class-wise and overall F1 scores of Bi-LSTM-CRF model using GloVe embeddings of 50 dimensions and 100 dimensions on test data

Figure 4.14 and Figure 4.15 show the overall F1 scores calculated using micro-averaging and macro-averaging and class-wise F1 scores of the Bi-LSTM-CRF NER model using GloVe embeddings of dimensions *50* and *100*. As per the concept of the curse of dimensionality, increase in number of features or dimension triggers the need for a bigger data set for better model performance. We analysed this by testing with a lower dimensional GloVe embedding, with a dimension of *50*. This model was also trained for epochs ranging from *10* to *50* using *4*-fold cross-validation and was repeated for five times. It was found to have the highest overall macro-average and micro-average F1 score of 0.88 when trained for *50* epochs and tested on the test data. This is slightly better than the overall micro and macro F1 score on the test data using *100-dimensional* GloVe embeddings of 0.87. Additionally, the class-wise F1 score using *50-dimensional* GloVe embeddings also shows a slight improvement for the majority of the classes with scores increasing from 0.84 to 0.86 for company name (25), 0.9 to 0.91 for meeting date (26), 0.81 to 0.83 for meeting address (27) except for (28) which gave a F1 score of 0.91 in comparison to 0.92 using the *100*-dimensional GloVe embeddings.

Chapter 5 Conclusions

In this thesis we compared different NER models based on Bi-LSTM-CRF with various word vectorization techniques like using Keras embedding layer, pre-trained GloVe embeddings and embeddings by the Elmo model. All models were compared on the same data set, and the base model used for all the models which is composed of Bi-LSTM-CRF was fixed to have the same configuration for comparing all three models. In addition, the output word embedding dimension used for comparison was set to *100* for Keras embedding layer and GloVe embeddings, while ELMo embeddings by default have an output dimension of *1024* which was kept unchanged. The effect of different word vectorization was analysed through the study. Thus, we managed to solve the problem of the presence of huge volumes of unstructured data at DNB by building an efficient NER system based on Bi-LSTM-CRF coupled with the best word vectorization technique. It was found that the Bi-LSTM-CRF NER model when coupled with *100-dimensional* GloVe embeddings gave the best performance. This was contrary to our hypothesis that contextual word embeddings would result in the best performing model. We proposed high dimensionality to be the underlying factor for the contradicting result and this was tested using a lower dimensional GloVe embedding which led to an improvement in performance. Hyperparameter tuning performed on the selected small search space did not lead to improvement in the model performance and it would be interesting to include more hyperparameters and increase the search space to tune the model in the future and push the performance scores further. It should be noted that the results of the study vary from the expected results based on previous researches conducted on various data sets like CoNLL 2003, 2010 i2b2/VA, Reaxys gold set and BioSemantics patent corpus. This could be due to the variation in the model architecture and the data set used. Hence an efficient NER found to perform well on the DNB data set might not be the most efficient one for other data sets. We can thus conclude that the models are dependent on the variations in the data set used. There do not exist one single best model that works best for all the data sets. In other words, there is no free lunch (NFL).

Future Work

The stochastic nature of deep learning models suggests that there exists variance in the performance score of the model, when the same model is trained on the same data set each time different performance score are obtained. The variance and standard deviation of the models were found to be a low value which we assume could be because of the low number of repetition of experiments performed for the study as well due to steps taken to handle randomness. The number of repeats were taken to be five for the thesis due to time and resource constraints. Randomness is introduced in deep learning models to introduce flexibility. It would be interesting to perform the experiments in the future without taking the steps to handle randomness and recording the distribution of scores for *30* to *100* repeats. In case of high variance, it would be interesting to explore the approach of using ensemble learning to reduce variance and make the model more robust [100]. Ensemble learning uses the concept of majority voting to make predictions. Members refer to the number of models in an ensemble. The steps to be followed are as follows:

1. Define the maximum number of members to be trained for ensemble learning; take *20,* for instance.
2. Train one member on the training data and make predictions for the test data and record the performance score of the model.
3. In the next step, train the second member and make predictions using prediction from step *2* and step *3* using majority voting, compute the final prediction for a two-member ensemble, evaluate the performance score of the ensemble.
4. Repeat step three for twenty members
5. Plot the performance score against the number of members in an ensemble.
6. Based on the plot, we can find a point at which the score no longer increases. The number of members corresponding to this point is chosen for the final ensemble model. Let five be the number of members, after which there is no improvement in the score.
7. Using the five-member ensemble, make the predictions, evaluate the performance score, repeat this for *30* times and find the variance between the scores, it would be observed that the variance between the scores is hugely reduced.

There exist numerous studies in the field of Named entity recognition using Bi-LSTM-CRF models as discussed in Sec. 1.3. Analysing the improvement in performance score of NER using a Bi-LSTM-CRF model due to the difference in architecture and additional features used

puts forward scope for future development, which would be interesting to explore. It could be observed that using character embeddings along with word embeddings results in an improvement in the model performance because of the improvement in the model's ability to capture morphological information related to words and hence this would be interesting to explore in the future [11, 12, 14, 15]. Study by Huang et al. [17] also reports that the performance of a Bi-LSTM-CRF NER with spelling features along with the word embeddings improved the model performance score. Previous research for clinical NER using clinical pre-trained ELMo word embeddings have also shown improvement in the performance of Bi-LSTM-CRF using general ELMo embeddings [23]. This could be an area that could be further explored by training an ELMo model from scratch, thus generating embeddings that are related to our data set instead of an ELMo model trained on a general data set.

Bibliography

1. Kiran Adnan , R.A., *An analytical study of information extraction from unstructured and multidimensional big data.* Journal of Big Data 2019. **6**.
2. Mónica Marrero, J.U., Sonia Sánchez-Cuadrado, and J.M.G.-B. Jorge Morato, *Named Entity Recognition: Fallacies, Challenges and Opportunities.* Computer Standards and Interfaces (CSI), 2012. **35**: p. 2.
3. Gantz J, R.D., *The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east*, in *IDC iView IDC Analyze Future*. 2012. p. 1–16.
4. Taylor, C. *Structured vs. Unstructured Data*. Big Data [Blog Post] 2018 2021.01.17]; Available from: https://www.datamation.com/big-data/structured-vs-unstructured-data.html.
5. Forsey, C. *What Is Semi-Structured Data?* Marketing 2019 2021-01-17]; Available from: https://blog.hubspot.com/marketing/semi-structured-data.
6. Eberendu, A.C., *Unstructured Data: an overview of the data of Big Data.* International Journal of Computer Trends and Technology, 2016. **38**: p. 46-50.
7. Chandershekher Joshi, L.D., Sandeep Saxena. *Leveraging Unstructured Text Data for Banks The Imperative for Mining Unstructured Data*. 2013 2021-01-11]; Available from: https://www.researchgate.net/publication/301821191_Leveraging_Unstructured_Text_Data_for_Banks_The_Imperative_for_Mining_Unstructured_Data.
8. Donghui Feng, G.B., Eduard Hovy *Extracting Data Records from Unstructured Biomedical Full Text* in *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. 2007.
9. Christopher Dozier, R.K., Marc Light,Arun Vachher, Sriharsha Veeramachaneni, Ramdev Wudali, *Named Entity Recognition and Resolutionin Legal Text.* Semantic Processing of Legal Texts, 2010.
10. Ku, P. *Big Data in Banking: Use Cases in 2020 and Beyond*. Informatica Blog [Blog Post] 2020 20201-01-17]; Available from: https://blogs.informatica.com/2020/04/13/big-data-banking-use-cases/.
11. Rofina Thomas, P.S. *Contract Intelligence (COiN) – A JP Morgan Case Study*. Artificial Intelligence News 2020 20201-01-20]; Available from: https://www.pye.ai/2020/11/18/contract-intelligence-coin-a-jp-morgan-case-study/.
12. Lei Xu, C.J., Jian Wang, Jian Yuan, *Information Security in Big Data: Privacy and Data Mining.* IEEE Access, 2014. **2**.
13. Matthew E. Peters, M.N., Mohit Iyyer, Matt Gardner,Christopher Clark, Kenton Lee, Luke 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*. 2018. Association for Computational Linguistics.

14. Hovy, X.M.a.E. *End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF*. in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016. Berlin, Germany: Association for Computational Linguistics.
15. Gurevych, N.R.a.J.E.-K.a.C.S.a.J.K.a.I. *GermEval-2014: Nested Named Entity Recognition with Neural Networks*. in *Workshop Proceedings of the 12th KONVENS 2014*. 2014.
16. Guillaume Lample, M.B., Sandeep Subramanian , Kazuya Kawakami , Chris 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*. 2016. Association for Computational Linguistics.
17. Zhiheng Huang, W.X., Kai Yu, *Bidirectional LSTM-CRF Models for Sequence Tagging.* ArXiv, 2015. **abs/1508.01991**.
18. Ismail El Bazi, N.L., *Arabic named entity recognition using deep learning approach.* International Journal of Electrical and Computer Engineering (IJECE), 2019. **9**: p. 2025-2032.
19. Mikolov, P.B.a.E.G.a.A.J.a.T., *Enriching Word Vectors with Subword Information.* Transactions of the Association for Computational Linguistics, 2017. **5**: p. 135-146.
20. Tjong Kim Sang, E.F., De Meulder, Fien. *Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition*. in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*. 2003.
21. Zhai, Z.a.N., Dat and Akhondi, Saber and Thorne, Camilo and Druckenbrodt, Christian and Cohn, Trevor and Gregory, Michelle and Verspoor, Karin. *Improving Chemical Named Entity Recognition in Patents with Contextualized Word Embeddings*. in *Proceedings of the 18th BioNLP Workshop and Shared Task*. 2019. Association for Computational Linguistics.
22. Saber A. Akhondi, A.G.K., Christian Tyrchan,Anil K. Manchala,Kiran Boppana,Daniel Lowe,Marc Zimmermann,Sarma A. R. P. Jagarlapudi,Roger Sayle,Jan A. Kors ,Sorel Muresan, *Annotated Chemical Patent Corpus: A Gold Standard for Text Mining.* PLoS One, 2014.
23. Tahmasebi, H.Z.a.I.P.a.A., *Clinical Concept Extraction with Contextual Word Embedding.* ArXiv, 2018. **abs/1810.10566**
.
24. Uzuner, O.a.S., Brett and Shen, Shuying and DuVall, Scott, *2010 i2B2/VA challenge on concepts, assertions, and relations in clinical tex.* Journal of the American Medical Informatics Association : JAMIA, 2011. **18**: p. 552-6.
25. Ratnaparkhi, A., *Maximum entropy models for natural language ambiguity resolution*. 1998, University of Pennsylvania.
26. Hamada A. Nayel, H.L.S., Hiroyuki Shindo,Yuji Matsumoto, *Improving Multi-Word Entity Recognition for Biomedical Texts.* International Journal of Pure and Applied Mathematics, 2018. **118**: p. 3.
27. Lance A. Ramshaw, M.P.M., *Text Chunking using Transformation-Based Learning* Third ACL Workshop on Very Large Corpora. MIT, 1995: p. 6-6.
28. Hiroki Nakayama, Y.T. *Doccano transformer*. 2020 [cited 2021 27.01.2021]; Available from: https://github.com/doccano/doccano-transformer.

29. PDF, A.F. *Types of PDFs*. ABBYY FineReader PDF n.d. 2021-01-26]; Available from: https://pdf.abbyy.com/learning-center/pdf-types/.
30. Lee, M. *pytesseract 0.3.6*. pypi 2020 2021-03-01]; Available from: https://pypi.org/project/pytesseract/0.3.6/.
31. Bhandari, A., *Build your own Optical Character Recognition (OCR) System using Google's Tesseract and OpenCV*, in *Analytics Vidya*. 2020.
32. tesseract-ocr. *Improving the quality of the output*. Tesseract documentation 2020 [cited 2021 04.02.2021]; Available from: https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html#page-segmentation-method.
33. Hiroki Nakayama, T.K., Junya Kamura ,Yasufumi Taniguchi ,Xu Liang. *Text Annotation Tool for Human*. 2018 2021-01-12]; Available from: https://github.com/doccano/doccano.
34. Sundheim, R.G.a.B. *Message Understanding Conference 6: A Brief History* in *The 16th International Conference on Computational Linguistics*. 1996.
35. Yoshua Bengio, R.D., Pascal Vincent , Christian Janvin and *A neural probabilistic language model.* The Journal of Machine Learning ResearchMarch 2003
, 2003. **3**: p. 1139-1139.
36. Rau, L.F., *Extracting company names from text*, in *Proceedings. The Seventh IEEE Conference on Artificial Intelligence Application*. 1991.
37. Georgios Petasis, A.C.p.i.C., P.V.p.i.V. , Georgios Paliouras profile imageGeorgios Paliouras, and V.K.p.i.K. , Constantine D Spyropoulos profile imageConstantine D. Spyropoulos, *Automatic adaptation of proper noun dictionaries through cooperation of machine learning and probabilistic methods.* Proceedings of 23rd annual international ACM SIGIR conference on Research and development in information
retrieval, 2000: p. 128-135.
38. Kripke, S.A., *Naming and Necessity*, ed. H.U. Press. 1980: Harvard University Press.
39. Chinchor, N., Robinson, P. *Appendix E: MUC-7 Named Entity Task Definition (version 3.5)*
. in *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*. 1998.
40. Sekine, S., K. Sudo, and S.S. Chikashi Nobata, H. Isahara and R. Grishman. *Extended Named Entity Hierarchy*. in *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*. 2002. Las Palmas, Canary Islands - Spain: European Language Resources Association (ELRA).
41. NIST. *NIST 2008 Automatic Content Extraction Evaluation (ACE08) - Official Results*. 2008 2021-03-25]; Available from: https://my.eng.utah.edu/~cs6961/papers/ACE-2008-description.pdf.
42. J.-D. Kim, T.O., Y. Tateisi, J. Tsujii, *GENIA corpus—a semantically annotated corpus for bio-textmining, Bioinformatics.* Bioinformatics, 2003. **19**(suppl_1): p. i180–i182.
43. Khurana, D., A.K. , and K.K. , and Sukhdev Singh, *Natural Language Processing: State of The Art, Current Trends and*
Challenges. ArXiv, 2017. **abs/1708.05148**.
44. HSIN–HSI CHEN, Y.W.D., SHIH–CHUNG TSAI, *Named entity extraction for information retrieval*, in *Computer Processing of Oriental Languages*. 1988. p. 75-85.
45. Anthony, B.B.a.H. *Improving Machine Translation Quality with Automatic Named Entity Recognition*. in *Proceedings of the 7th International EAMT workshop on MT and*

other language technology tools, Improving MT through other language technology tools, Resource and tools for building MT}at EACL 2003. 2003.
46. Chikashi Nobata, S.S., H. Isahara and R. Grishman, *Summarization System Integrated with Named Entity Tagging and IE pattern Discovery*

, in *Proceedings of the Third International Conference on Language Resources and Evaluation LREC'02*. 2002, European Language Resources Association (ELRA). p. 742--1745.
47. Molla Aliod, D.a.Z., Menno and Smith, Daniel. *Named entity recognition for question answering*. in *ALTA*. 2006.
48. Urbano, J., *Information Retrieval Meta-Evaluation:Challenges and Opportunities in the Music Domain.* International Society for Music Information Retrieval Conference, 2011: p. 597-602.
49. Wenming Huang, D.H., Zhenrong Deng & Jianyun Nie *Named entity recognition for Chinese judgment documents based on BiLSTM and CRF.* EURASIP Journal on Image and Video Processing volume 2020.
50. Bisong, E., *Google Colaboratory* in *Building Machine Learning and Deep Learning Models on Google Cloud Platform* 2019, Apress, Berkeley, CA
. p. 59-64.
51. Martín Abadi, A.A., Paul Barham, Eugene Brevdo,, et al. *Tokeniser -TensorFlow: Large-scale machine learning on heterogeneous systems*. TensorFlow documentation 2015 2021-04-05]; Available from: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer.
52. P., S., *Natural Language Understanding.* , in *Artificial Neural Networks with TensorFlow 2*. 2021, Apress, Berkeley, CA.
53. Sethi, A. *One-Hot Encoding vs. Label Encoding using Scikit-Learn*
. Analytics Vidya 2020; Available from: https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/Chigozie
54. Yann LeCun, Y.B., Geoffrey Hinton, *Deep Learning.* Nature, 2015. **521**.
55. Sebastian Rashka, v.M., *Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*, in *Python Machine Learning*. 2019, Packt Publishing: BIRMINGHAM - MUMBAI. p. 590-592.
56. Chigozie Enyinna Nwankpa, W.I., Anthony Gachagan, and Stephen Marshall. *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning*. in *Conference: 2nd International Conference on Computational Sciences and Technology, (INCCST)* 2020. Jamshoro, Sindh Pakistan.
57. A Vehbi Olgac, B.K., *Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks.* International Journal of Artificial Intelligence And Expert Systems, 2011. **1**: p. 111-122.
58. mvs, C. *Activation Functions : Why "tanh" outperforms "logistic sigmoid"?* [Blog Post] 2019 [cited 2021 2021-05-17]; Available from: https://medium.com/analytics-vidhya/activation-functions-why-tanh-outperforms-logistic-sigmoid-3f26469ac0d1#:~:text=But%2C%20always%20mean%20of%20tanh,zero%20when%20compared%20to%20sigmoid.&text=It%20can%20also%20be%20said,better%20than%20sigmoid%20(logistic).
59. Hochreiter, S., *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis,.* 1991.

60. Vinod Nair, G.E.H., *Rectified linear units improve restricted boltzmann machines*, in *Proceedings of the 27th International Conference on International Conference on Machine Learning*
. 2010: Haifa, Israel. p. 807–814.
61. Gupta, D., *Fundamentals of Deep Learning – Activation Functions and When to Use Them?*, in *Analytics Vidya*. 2020: Analytics Vidya.
62. Brownlee, J. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Machine learning Mastery 2019 [cited 2021; Available from: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.
63. Alex Krizhevsky, I.S., Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional*
Neural Networks, in *Advances in Neural Information Processing Systems*, F.P.a.C.J.C.B.a.L.B.a.K.Q. Weinberger, Editor. 2012, Curran Associates, Inc.
64. contributors, W. *Sequence*. 2021 [cited 2021 2021-02-07]; Available from: https://en.wikipedia.org/w/index.php?title=Sequence&oldid=1024064161.
65. David E. Rumelhart, G.E.H., Ronal J. Williams, *Learning internal representations by error propagation.*
. MIT Press, 1986. **1**: p. 318–362.
66. Peilu Wang, Y.Q., Frank K. Soong, Lei He, Hai Zhao, *Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network.* CoRR, 2015. **abs/1510.06168**.
67. Yann N. Dauphin, A.F., Michael Auli, David Grangier. *Language Modeling with Gated Convolutional Networks*. in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. 2017. JMLR.org.
68. Werbos, P.J., *Backpropagation through time: what it does and how to do it.* IEEE Xplore, 1990. **78**: p. 1550-1560.
69. Mike Schuster, K.K.P., *Bidirectional Recurrent Neural Networks.* IEEE TRANSACTIONS ON SIGNAL PROCESSING,, 1997. **45**: p. 2673-2681.
70. Nils Reimers , I.G. *Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging*. in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017.
71. CreateMoMo. *CRF Layer on the Top of BiLSTM* CreateMoMo 2017 2021-01-24]; Available from: https://createmomo.github.io/2017/09/23/CRF_Layer_on_the_Top_of_BiLSTM_2/.
72. Łukasz Augustyniak, T.K., Przemysław Kazienko, *Aspect Detection using Word and Char Embeddings with (Bi) LSTM and CRF*, in *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering*. 2019. p. 43-50.
73. John Lafferty, A.M., Fernando C.N. Pereira. *Conditional Random Fields: Probabilistic Models*
for Segmenting and Labeling Sequence Data. in *Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001)*. 2001.
74. Hinton, G., *Neural Networks for Machine Learning - Lecture 6a - Overview of mini-batch gradient*
descent. 2012.
75. Graves, A., *Generating Sequences With Recurrent Neural Networks.* CoRR, 2013. **abs/1308.0850**

.
76. Pai, A. *An Essential Guide to Pretrained Word Embeddings for NLP Practitioners*. Analytics Vidhya 2020 [cited 2021 2021-01-16]; Available from: https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/.
77. Jeffrey Pennington, R.S., Christopher D. Manning. *GloVe: Global Vectors for Word Representation*. in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014. Association for Computational Linguistics.
78. Bruno Taillé, V.G., Patrick Gallinari, *Contextualized Embeddings in Named-Entity Recognition: An Empirical Study on Generalization*, in *Advances in Information Retrieval*. 2020, Springer, Cham. p. 383-391.
79. Martín Abadi, A.A., Paul Barham, Eugene Brevdo,, et al. *Masking and padding - TensorFlow: Large-scale machine learning on heterogeneous systems*. TensorFlow documentation 2015 [cited 2021 05.04.2021]; Available from: https://www.tensorflow.org/guide/keras/masking_and_padding.
80. Keras. *Keras lambda layer*. Keras documentation n.d 2021-04-07]; Available from: https://keras.io/api/layers/core_layers/lambda/#:~:text=The%20Lambda%20layer%20exists%20so,simple%20operations%20or%20quick%20experimentation.&text=Lambda%20layers%20are%20saved%20by,which%20is%20fundamentally%20non%2Dportable.
81. Keras. *Layer weight initializers*. Keras documentation n.d. [cited 2021 02.02.2021]; Available from: https://keras.io/api/layers/initializers/.
82. Pelletier, C.a.W., Geoffrey and Petitjean, François, *Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series.* Remote Sensing, 2019. **11**.
83. Google. *ELMo V2*. 2021-03-15]; Available from: https://tfhub.dev/google/elmo/2.
84. Brownlee, J. *Difference Between a Batch and an Epoch in a Neural Network*. Machine learning mastery 2018 20201-02-01]; Available from: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.
85. David Nadeau, S.S., *A survey of named entity recognition and classification.* Lingvisticae Investigationes, 2007. **30**: p. 3-26.
86. Shmueli, B., *Multi-Class Metrics Made Simple, Part II: the F1-score*. 2019, towardsdatascience.com.
87. Nakayama, H. *seqeval: A Python framework for sequence labeling evaluation*. chakki-works 2018; Available from: https://github.com/chakki-works/seqeval.
88. Hajic, J.S.a.M.S.a.J., *Neural Architectures for Nested NER through Linearization.* arXiv e-prints, 2019: p. 5326-5331.
89. Brownlee, J. *A Gentle Introduction to k-fold Cross-Validation*. Machine learning mastery [Blog post] 2020 03.08.2020 [cited 2021; Available from: https://machinelearningmastery.com/k-fold-cross-validation/.
90. Gurevych, N.R.a.I., *Optimal Hyperparameters for Deep LSTM-Networks for Sequence Labeling Tasks.* CoRR, 2017. **2**.
91. Benajiba, Y., *Arabic named entity recognition.* 2009.
92. Semeniuta, S.a.S., Aliaksei and Barth, Erhardt. *Recurrent Dropout without Memory Loss*. in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. 2016. The COLING 2016 Organizing Committee.

93. Ghahramani, Y.G.a.Z., *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks.* Advances in Neural Information Processing Systems, 2015. **29**.
94. Diederik P. Kingma, J.B., *Adam: A Method for Stochastic Optimization.* CoRR, 2015. **abs/1412.6980**.
95. Dozat, T., *Incorporating Nesterov Momentum into Adam.* ICLR Workshop, (1), 2016.
96. Yang, Y.a.L., Xin, *A Re-Examination of Text Categorization Methods.* Proceedings of the 22nd SIGIR, New York, NY, USA, 2003: p. para 8.
97. Bellman, R., *Dynamic Programming*. Vol. 153. 1957: Princeton : Princeton University Press, 1957.
98. Piccardi, R.C.a.E.Z.B.a.M. *Bidirectional LSTM-CRF for Clinical Concept Extraction*. in *The COLING 2016 Organizing Committee*. 2016.
99. Okewu E., A.P., Sennaike O, *Experimental Comparison of Stochastic Optimizers in Deep Learning.*, in *Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science*. 2019, Springer International Publishing. p. 704--715.
100. Brownlee, J. *How to Develop an Ensemble of Deep Learning Models in Keras*. 2018 2021-04-23]; Available from: https://machinelearningmastery.com/model-averaging-ensemble-for-deep-learning-neural-networks/.

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway