

DEPENDENCY ENCODING FOR RELATION EXTRACTION

FATHIMA NIHATHA ABDUL LATHIFF

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

JUNE 2022

© FATHIMA NIHATHA ABDUL LATHIFF, 2022

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Fathima Nihatha Abdul Lathiff**

Entitled: **Dependency Encoding for Relation Extraction**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
(Dr. René Witte)

_____ Examiner
(Dr. Olga Ormandjieva)

_____ Examiner
(Dr. René Witte)

_____ Thesis Supervisor
(Dr. Sabine Bergler)

Approved by _____

Dr. Leila Kosseim
Chair of Department or Graduate Program Director

Mourad Debbabi, Ph.D., Dean
Faculty of Engineering and Computer Science

Abstract

Dependency Encoding for Relation Extraction

Fathima Nihatha Abdul Lathiff

The surge in information in the form of textual data demands automated systems to extract structured information from unstructured data. Relation extraction plays a key role in the process, with the aim of extracting semantic relations between entities in a text. Since dependency parse trees are capable of capturing the grammatical structure of sentences, this thesis experiments with different encodings of the dependency parse tree to distinguish different semantic relationships. Experiments are conducted on three different data sets that vary in domain and complexity and experimented with varying encoding schemas that can be grouped into two. The first group focuses on encoding the structure of the dependency parse tree with a Deep Graph Convolution Neural Network (DGCNN). The second group focuses on encoding the linguistic features obtained from the dependency parse tree with classical machine learning models such as Random Forest, Support Vector Machine, and Feed-Forward Network, and deep models such as BERT and Transformer encoder stack. The objective of this thesis is not to achieve state-of-the-art (SOTA) performance, rather to evaluate how dependency parse tree based linguistic features perform on different encoding schemas, including deep transformer-based models, on the relation extraction task. The results of the experiments show that these features on certain data sets being less computationally demanding are competitive for complex language models such as BERT, and incorporating them externally to BERT improves the performance rather than confounding.

Acknowledgments

Completing a Master program for many can be just another step in life. However, for me, completing a Master's program in Canada is a game changer. The journey from beginning to end was not easy and I would not have been able to do it without the help of many.

First, I would like to convey my sincere gratitude to my thesis supervisor, Dr.Sabine Bergler, for giving me the opportunity to convert myself from a course-based student to a thesis student and work under her supervision. Although everything was new to me, including NLP, she has shown me immense support and patients to guide me to completion. Late-night paper submissions and disapproval over ideas were hectic then, but are cherished memories and valuable lessons learned that will last forever with me. Thank you so much for being there at a crucial time in my life as a supervisor.

However, without the help of Dr.Leila Kosseim, I would not have managed to enroll into the NLP course at the very first semester, which led me into becoming a thesis student. Also, on her recommendation, I became a teaching assistant for Dr.René Witte for many semesters, which helped me to financially stabilize myself through my graduate studies. I am very grateful to both Dr.Kosseim and Dr.Witte, for all the timely assistance throughout this journey.

Studying as an international student is not cheap and the burden of student loans will last a long time. But I managed to complete without any financial stress and all the thanks should go to my elder brother, Nabil Lathiff, for sponsoring me and supporting me in every way possible. I am also grateful to my mom Nooriya Lathiff for constantly checking on me for my well-being and my dad Dr.Abdul Lathiff for sharing his valuable experiences on how to process through graduate studies.

The best way to gain knowledge is to share knowledge and I have seen this extensively in my lab. I am sincerely grateful for all the valuable discussions, insights and unwavering support of all my colleagues at CLaC Lab Parsa Bagherzadeh, Mingyou Sung, Narjesossadat Tahaei, Nadia Sheikh, Zhanfan Zhou, and Harsh Verma. Thank you very much for your

motivational words and support throughout my graduate program. Although I did not have the opportunity to work with them directly in the lab, I extend my special thanks to Nadia Bilal, who morally and physically supported me in many ways by constantly checking on me, and Sunanda Bansal for whom I do not have words to express how grateful I am. Although I knew her by name and may have uttered a few words in the past, during the process of writing this thesis, if not for her relentless support, I would not have come this far. Thank you so much, for all your patients and the hours and hours spent listening to me and helping me formulate my ideas.

It is luck or blessing to have very kind and thoughtful people around you, and I always managed to have really good friends that I can count on. Many thanks to all my friends Adriana Gonzalez, Akshat Bisht, Nikitha Bangera, Reethu Navale, Jaiganesh Varatharaju, and Shabnam Hassan for always being there whenever I feel down. I also would like to express my gratitude to the Nearu Martial Arts group for being there as a family and keeping me physically fit to get through the program. Special thanks to Prasanna Shanmugarajah and Abiramy Shanmugarajah for being my life support and guiding me in all possible ways to get me started with my Masters' program. If not for their guidance and immense support, I would not be here today.

I saved the best for last, Pavel Khloponin, to whom I am deeply indebted. If not for his patients, guidance and moral support, I don't think I would have managed to come this far. Thank you for accepting me for who I am, thank you for holding me whenever I go through a mental breakdown, and thank you very much for patiently listening to all kinds of whining and complaints.

This thesis would not have been completed without the help of you all. Thank you very much.

Contents

List of Figures	ix
List of Tables	xii
1 Introduction	1
1.1 Drug-Drug-Interactions (DDI)	4
1.2 SemEval-2010 task8: Multi-way classification of semantic relations between pairs of nominals (Sem10)	5
1.3 MeasEval: Counts and Measurements (MeasEval)	8
2 Background	12
2.1 Relationship Extraction (RE)	12
2.2 Classification Models	14
2.2.1 Feed-Forward Neural Network	14
2.2.2 Support Vector Machines (SVM)	15
2.2.3 Convolutional Neural Networks (CNN)	16
2.2.4 Transformers	18
2.3 Linguistic features	21
2.3.1 Tokenization	22
2.3.2 Part-of-Speech (POS) tagging	23
2.3.3 Dependency Parsing	23
2.4 Feature Representations	24
2.4.1 One-hot-encoding	25
2.4.2 Label encoding	25
2.4.3 Multi-hot encoding	26
2.4.4 Word Embedding	26
2.4.5 Generalized Language Models	28
2.5 Works related to Relationship Extraction	33

3	Dependency parse trees and Graph Neural Networks	35
3.1	Graph Neural networks (GNN)	36
3.1.1	Deep Graph Convolution Neural Network (DGCNN)	37
3.2	Preprocessing text	43
3.2.1	Pre processing pipeline	43
3.3	Constructing Information Matrix (IM)	44
3.3.1	Target entity pair extraction and class labeling for DDI	45
3.3.2	Target entity pair extraction and class labeling for Sem10	46
3.3.3	Target entity pair extraction and class labeling for MeasEval	47
3.3.4	Extracting Subtree encompassing target entities and representing features	54
3.3.5	Final representation of IM	62
4	Experiments and Analysis	64
4.1	Evaluation metrics	64
4.2	GNN based architecture: DGCNN	66
4.2.1	Original DGCNN ($DGCNN_{org}$)	66
4.2.2	Modified System ($DGCNN_{mod}$)	66
4.3	Encoding Information matrix	68
4.4	Classic baseline algorithms	70
4.4.1	Random Forest (RF)	70
4.4.2	Support Vector Machines (SVM)	71
4.4.3	Feed-Forward Neural network with Non linear activation (FF)	71
4.5	Pretrained baseline model - BERT	72
4.6	Concatenating Bert output with external representations	74
4.6.1	Concatenating IM_{row_concat} as an external feature vector (BERT_IM)	74
4.6.2	Concatenating BERT and Convolved IM (BERT_IM $_{conv}$)	75
4.6.3	Concatenating BERT and DGCNN (BERT_DGCNN)	76
4.7	Six layer Transformer Encoder Stacks (6TS)	82
4.8	Overall comparison	84
4.9	Re-evaluating the IM matrix	91
4.9.1	Dependency-based IM (IM_{dep}) versus Sentence-based IM (IM_{sent})	92
4.9.2	Dependency path versus Dependency governor	93
5	Conclusion and Future Work	94
	Bibliography	97

Appendix A Ablation study on linguistic feature representation	106
Appendix B Official evaluation scheme of the MeasEval task	108

List of Figures

1.1	Dependency parse tree of Ex 1; highlighted relationship indicate the shortest path between the two entities <i>Fluvoxamine</i> and <i>alostron</i>	3
2.1	Illustration of a fully connected feed-forward neural network	15
2.2	Hyperplane and support vectors in SVM	16
2.3	Transformation of features to higher dimension with kernel trick.	16
2.4	1D Convolution Neural Network	17
2.5	Attention weights associated with source words (English) to target words (French) in English-French machine translation task (Bahdanau, Cho, Hyun, & Bengio, 2015)	19
2.6	Different memory activation indicating the influence of memory words (blue) on the target word (red). Intensity of blue indicates the degree of memory activation. (Cheng, Dong, & Lapata, 2016)	20
2.7	Illustration of Transformer Encoder architecture (Vaswani et al., 2017)	20
2.8	Unicode tokenization	22
2.9	Wordpiece tokenization	23
2.10	POS tagging of Ex 6 from Stanford Parser (Klein & Manning, 2003)	23
2.11	A typed dependency parse from the Stanford Parser (de Marneffe, MacCartney, & Manning, 2006)	24
2.12	Tree-view of dependency parsed sentence shown in Figure 2.11	24
2.13	Architecture of Word2Vec Continuous Bag-of-words (CBOW) model	27
2.14	Architecture of Word2Vec skip-gram model	28
2.15	Input embeddings of BERT consisting of token embeddings, segmentation embeddings and positional embeddings. Source (Devlin, Chang, Lee, & Toutanova, 2019)	31
2.16	Training parameters added to the base model for fine-tuning BERT for different downstream task. Source (Devlin et al., 2019)	32
3.1	Graphs and adjacency matrices	38

3.2	Representation of an Adjacency matrix and corresponding Information matrix for entity pair (<i>stress, divorce</i>)	39
3.3	Illustration of DGCNN architecture; Source: M. Zhang, Cui, Neumann, and Chen (2018)	40
3.4	Architecture of 1D convolutional neural network in DGCNN	42
3.5	Tokenized and POS tagged sample sentence from the MeasEval data set	49
3.6	Gold annotation and dependency-parse tree representation of the sentence in Figure 3.5	53
3.7	Dependency subtree of a Sem10 sample, $(E_1, E_2) = (stress, divorce)$; Dependency path $p(E_1, t_i)$ generated for all nodes / tokens within the subtree with respect to E_1	57
3.8	Muli-hot encoding of dependency path of $p(stress, divorce) = (nsubj\ nmod\ nmod)$	58
3.9	Concatenated node2vec encoding of dependency path $p(stress, divorce)$; $lp = 6$ and $de = 128$	58
3.10	Directed acyclic graph with a maximum degree of 1.	60
3.11	First-order transition probabilities	60
3.12	Illustration of random walk procedure in node2vec. The walk just transitioned from u_2 to v and is now evaluating its next step out from v . Edge label indicates search biases α ; yellow line indicates the in-out edge, and red line indicates the return edge.	62
3.13	Detailed view of the feature vector in IM	62
4.1	Row concatenation approach to construct one-dimensional vector	69
4.2	Column aggregation approach to construct one-dimensional vector	70
4.3	BERT language model combined with a FF network as classifier to perform RE task. Dotted lines indicate the back propagation path. BERT input constructed with 3 segments; E_1 , E_2 and tokenized sentence, with [SEP] marker after every segment.	73
4.4	BERT- IM architecture; IM_{row_concat} concatenated with final hidden representation of [CLS] token from BERT as used as input to the FF classifier. Dotted lines indicate the back propagation path of the error.	75
4.5	BERT- IM_{conv} architecture; concatenation of final hidden representation of [CLS] token from BERT with output of 1D convolution performed on IM_{row_concat} prior to feeding to a FF classifier. Dotted lines indicate the back propagation path of the error to fine-tune the weight parameters of convolution network and BERT.	76

4.6	BERT_DGCNN architecture; concatenation of final hidden representation of [CLS] token from BERT with the intermediary representation obtained after the 1-D convolution stage of DGCNN, prior to using it as input to a FF classifier. Dotted lines indicate the back propagation path of the error to fine-tune the weight parameters of BERT and DGCNN.	77
4.7	Recall (left) and Precision (right) values of BERT and BERT_IM model for each class in the Sem10 data set; OTH:Other, C-E: Cause-Effect, I-A: Instrument-Agency, P-P: Product-Producer, C-C: Content-Container, E-O: Entity-Origin, E-D: Entity-Destination, C-W: Component-Whole, M-C: Member-Collection, M-T: Message-Topic	78
4.8	6 layer Transformer encoder stack with <i>IM</i> as input	83
4.9	Line chart illustrating F1 measure of different encoding schema for each class in Sem10 data set	86
B.1	Format of a .tsv output file expected by the MeasEval evaluation script. <i>docId</i> : the document id of the evaluated text; <i>annotSet</i> : the annotation set an annotation type belongs to; <i>annotType</i> : the annotation type of each detected entity; <i>startOffset</i> , <i>endOffset</i> : the starting and ending offset respectively of the spans of the entities identified for each annotation type; <i>text</i> : resembles the text withing the span of the offsets identified; <i>other</i> : includes additional details such as relationship type, if the annotation type is any other than quantity, and for quantity this includes the units and modifier if identified any.	109

List of Tables

1.1	Basic statistics on the DDI	5
1.2	Basic statistics on Sem10 data set	7
1.3	Basic statistics on MeasEval data set	11
2.1	Activation functions	14
2.2	Sample context-target pairs generated, with context window size 2 for sentence: “ <i>The more you know, the more you realize you know nothing</i> ” . . .	27
3.1	Training data: Frequencies of Annotation types	48
3.2	Frequencies of different composition of Annotation Sets	48
3.3	Word count within spans for each annotation type. Spans are split by space to obtain an approximate word count in given gold data.	50
3.4	Continuation of Table 3.3... Word count within spans for each annotation type. Spans are split by space to obtain an approximate word count in given gold data.	51
3.5	Class labels assigned to token pairs generated for base_CD “2” on Figure 3.5	54
3.6	Representation techniques used to represent linguistic features in <i>IM</i> ; bold text indicate the chosen representation technique for each feature.	63
4.1	Default parameter setting of each stage in the DGCNN architecture defined by the authors (M. Zhang & Chen, 2018)	67
4.2	Precision(P), Recall(R) and F1-measure(F1) results on all 3 data sets with original DGCNN ($DGCNN_{org}$) and parameter modified DGCNN ($DGCNN_{mod}$). 67	
4.3	Precision(P), Recall(R) and F1-measure(F1) results on all 3 data sets with baseline models RF, SVC and FF with 2 different encoding of the IM (IM_{col_aggr} and IM_{row_concat})	72
4.4	Precision(P), Recall(R) and F1 measure(F1) results on all 3 data sets with BERT based architectures	77
4.5	Confusion matrix of BERT on Sem10 data set	79

4.6	Confusion matrix of BERT_IM on Sem10 data set	79
4.7	BERT confusion matrix on DDI	80
4.8	BERT_IM confusion matrix on DDI	80
4.9	Comparison results of BERT based and DGCNN based models	82
4.10	Precision (P), Recall (R), and F1 measure (F1) on all 3 data sets based on 6TS. The ‘with’ and ‘no’ positions indicate whether positional encoding is embedded in the 6TS input.	83
4.11	Summary of the results of the best performing configuration of different encoding systems	84
4.12	Distribution of classes in Sem10 data set	87
4.13	Ranking of Sem10 classes based on the F1 score for each encoding system	88
4.14	Summary of classes confused with other classes based on confusion matrix of BERT (Table 4.5) with examples; C-E: Cause-Effect, I-A: Instrument-Agency, P-P: Product-Producer, C-C: Content-Container, E-O: Entity-Origin, E-D: Entity-Destination, C-W: Component-Whole, M-C: Member-Collection, M-T: Message-Topic	89
4.15	Precision(P), Recall(R) and F1-measure(F1) of all 3 data sets to evaluate the impact of PubMed ELMo (FF_{pubmed}) versus Original ELMo (FF_{org}), and token embedding (FF (Token)) versus all features (FF (All)) in a FF network	91
4.16	Precision (P), Recall (R), and F1 measure (F1) of Sem10 and DDI data set to evaluate sentence-based IM (IM_{sent}) versus dependency-based IM (IM_{dep}) on the DGCNN and BERT_IM encoding systems	92
4.17	Precision (P), Recall (R) and F1 measure (F1) of Sem10 and DDI data sets to evaluate the dependency path encoding (dep_path) against dependency governor relationship (dep_gov)	93
A.1	Partial results of an ablation study conducted on different representation of linguistic features on MeasEval data set on DGCNN system	107

Chapter 1

Introduction

In bio-medicine, knowing the impact of introducing a new drug in the presence of another is crucial to avoid any fatal consequences, in particular, when the reaction is positive. Knowledge repositories with information on drugs and their behavior in the presence of others make information retrieval on drugs easier, rather than having to go through hundreds of pages of medical documents. However, the need for a structured representation of knowledge possibly in the form of knowledge graphs is not limited to the biomedical domain. Rather, it is of crucial significance to many NLP applications such as sentiment analysis, summarization, structured search, question answering and sentiment analysis.

Due to technological advancement and with the introduction of the Web in 1983, the surge of electronic text in the form of news wires, blogs, emails, tweets, publications, etc., started to exponentially increase in terms of size and in heterogeneity. This made it difficult for people to stay up-to-date with trending changes or access prior knowledge. Therefore, to ease the process of acquiring up-to-date information from many sources published daily, there exists a need for automated extraction of structured information from unstructured electronic text documents. This process is known as Information Extraction (IE) in the field of Natural Language Processing (NLP). [Grishman \(1997\)](#) defined IE as follows:

“The process of IE is the identification of instances of a particular class of events or relationships in a natural language text, and the extraction of the relevant arguments of the event or relationship. Information extraction therefore involves the creation of a structured representation (such as a database) of selected information drawn from the text.”

([Grishman, 1997](#))

According to [Jurafsky and Martin \(Jurafsky & Martin, 2009\)](#), IE is a pipeline-like

process that consists of several stages such as: Named Entity Recognition (NER), Reference Resolution, Relationship Detection and Classification / Relationship Extraction (RE), Event Detection and Classification, Temporal Expression Recognition and Temporal Analysis, and Template Filling. Although these stages are part of the pipeline, each stage on its own is a research field in NLP. Since the main objective of this thesis is to identify the relationship between two entities of interest given a text, I focus on the RE (relationship extraction) stage in the IE pipeline.

Relationship extraction refers to a step in IE that extracts relationships between entities of interest based on the language cues found in the text. However, the type of relationship (and the entities) is highly dependent on the context and objective of the task. In other words, there is no single universal set of relationship types defined; instead, the relationship varies task-to-task, relying on the objective of what to achieve or extract from a corpus. The following are examples of different types of relationship types (and entity types) extracted from different corpora (the underlined text in the following examples denotes different types of entity identified).

Ex 1: “Fluvoxamine increased mean aloseptron plasma concentration (AUC) approximately 6-fold and prolonged the half-life by approximately 3-fold.”

Relation: (fluvoxamine, alosetron) = TRUE - DRUG-DRUG INTERACTION

(SemEval 2013 - DDI dataset ([Segura-Bedmar, Martínez, & Herrero-Zazo, 2013](#)))

Ex 2: “Financial stress is one of the main causes of divorce”

Relation: (stress, divorce) = CAUSE-EFFECT

(SemEval 2010 - Relation Extraction dataset ([Hendrickx et al., 2010](#)))

Ex 3: “Over the range tested, the deposition rate of perchloric acid increases by around six orders of magnitude.”

Relation: (deposition rate, around six orders of magnitude) = HAS_QUANTITY

Relation: (perchloric acid, deposition rate) = HAS_PROPERTY

Relation: (increases, deposition rate) = QUALIFIES

(SemEval 2021 - MeasEval ([Harper et al., 2021](#)))

As illustrated in examples Ex 1 - 3, the relation extraction task is not confined to defined definitions. However, based on the grammatical structure of a sentence, one can try to predict whether there exists any type of relationship between the two entities of interest. For example, based on Ex 1, given the two entities *Fluvoxamine* and *aloseptron*, we can predict that there is an interaction between the two by observing words such as *increased*

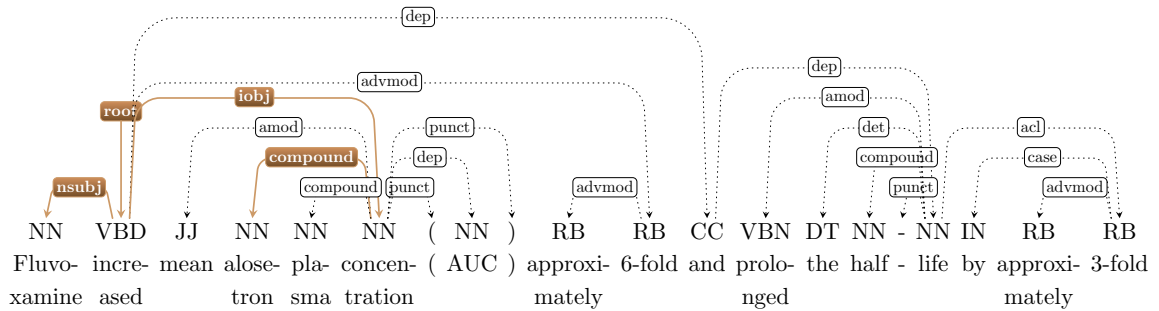


Figure 1.1: Dependency parse tree of Ex 1; highlighted relationship indicate the shortest path between the two entities *Fluvoxamine* and *aloston*

mean ... concentration. To extract such necessary information one can use dependency parse trees which are known to capture the grammatical structure of a sentence. Figure 1.1 shows the dependency parse tree of Ex 1. Based on the figure, it can be seen that *Fluvoxamine* and *aloston* are connected through the word *increased*. This inherently indicates the influence of the noun subject (nsubj) *Fluvoxamine* on the object *aloston*, which is a compound of the indirect object (iobj), *concentration*. Therefore, considering the nature of the RE task and the capability of dependency parse trees on capturing semantic relations between entities, this thesis focuses on different ways to encode dependency parse tree using varying neural models to perform the RE task.

With time, change in the nature of text and size lead to the growth in deep learning approaches. This in return paved the path for advanced problem solving in NLP, reduced the focus from using static linguistic features which were heavily used in early NLP research. Furthermore, deep learning models such as BERT (Devlin et al., 2019) are designed in such a way that there is no one direct way to encode linguistic features externally with its input. Therefore, this thesis more precisely focuses on 2 things. First, encoding linguistic features extracted from the dependency parse tree along with features representing the parse tree to evaluate whether these features are suitable for the RE task. Second, using these linguistic features as input to different encoding schemas including deep transformer encoder-based models to evaluate the impact of these features on these models' performance. To evaluate the generality of different approaches across text from varying domains, this thesis focuses on three different data sets that vary in complexity and in the objective of the task. The following sections will focus on a brief description of different datasets in increasing order of complexity.

1.1 Drug-Drug-Interactions (DDI)

Objective of the task: Medicines help us lead a better life and stay healthy. However, the wrong dose or combination of different drugs can cause fatal results. Since negative DDI can have dangerous therapeutic effects, the detection of DDI is an important field of research for both patient safety and proper cost control of health care. Therefore, many studies on drug-drug interactions have gained interest and are frequently reported in clinical pharmacology journals and technical reports, leading to the challenge task DDI Extraction-2011 (Segura-Bedmar, Martinez, & Sanchez-Cisneros, 2011) followed by DDI Extraction-2013 (Segura-Bedmar et al., 2013). These tasks were created with the objective of advancing state-of-the-art text mining techniques to facilitate drug identification and the extraction of drug-drug interaction information from the biomedical literature.

Subtasks: Unlike the DDI Extraction-2011 (Segura-Bedmar et al., 2011), DDI Extraction-2013 (Segura-Bedmar et al., 2013) is made up of 2 subtasks:

- (1) Task 1: Identification and classification of pharmacological substances.
- (2) Task 2: Extraction of drug-drug interaction and its type

The competition was held in such a way that the two subtasks were independent of each other. Drug detection (Task 1) is not necessary for extraction of the relation (Task 2) since the drugs were explicitly annotated. Since the main focus of this thesis was on the extraction of relationships, I focused only on Task 2, detecting whether there exists an interaction between drugs or not.

Dataset The semantically annotated Drug-Drug Interaction (DDI) corpus labels documents for drug-drug interactions in form of XML. Each file consists of a number of sentences with drugs and their interactions (TRUE or FALSE) annotated. Drugs that have a positive interaction with another will have *type* interaction as additional information; the type can be one of the four types of interactions that drugs could have between them: Advise, Effect, Mechanism, and Interaction. However, in this thesis, the DDI data set is treated as a binary classification task to predict true or false values based on whether or not the two drugs have an interaction. Ex 1 mentioned above is a sample data set from the DDI corpus that illustrates the annotated drugs and their relationship. However, unlike the demonstrated example, sentences in the DDI can contain multiple drug entities within a sentence, where relationships should be predicted for all permutations of drug combinations (even if the drug name is repeated multiple times within the sentence), as shown in Ex 4.

Ex 4: “Digoxin¹: When multiple doses of atorvastatin and digoxin² were co-administered, steady-state plasma digoxin³ concentrations increased by approximately 20%.”

Relations: FALSE (digoxin¹, atorvastatin),
 FALSE (digoxin¹, digoxin²),
 FALSE (digoxin¹, digoxin³),
 TRUE (atorvastatin, digoxin²),
 FALSE (atorvastatin, digoxin³),
 FALSE (digoxin², digoxin³)

As illustrated in Ex 4, creating all permutations of drug-drug pairs results in a high number of data points falling into the negative class, and repeated drug names have an ambiguous relationship with other drugs depending on the position of the drug name within the sentence. This strongly calls for models to focus on the relationship between words considering sentence structure rather than on pre-trained token representations.

The DDI corpus consists of 1,072 documents, of which 714 documents were assigned as training data, and the remaining were treated as test data sets. The basic statistics of the corpus are shown in Table 1.1.

Table 1.1: Basic statistics on the DDI

	Training	Testing
Total number of sentences	6976	1964
Total number of entities	14765	3726
Total number of pairs	27792	6657
Sentences without entities	1416	554

1.2 SemEval-2010 task8: Multi-way classification of semantic relations between pairs of nominals (Sem10)

Objective of the task: The SemEval-2010 task 8 (Hendrickx et al., 2010) is focused on the extraction of semantic relations between pairs of nominal given a sentence. Rather targeting a specific domain, this corpus focuses on extracting generic relationships of a broad variety and of practical interest that can be observed in English texts. The organizers formulated this task as a multi-way classification task where for each example, one class has to be chosen from a set of nine defined relationships and OTHER if no relationship is detected, in addition to the mapping from nouns to arguments. The nine relationships are defined as follows with examples:

- (1) CAUSE-EFFECT (CE): Represents an event or an object leading to an effect

In chemical lasers the <e1> inversion </e1> is produced by a chemical <e2> reaction </e2>.

- (2) INSTRUMENT-AGENCY (IA): Represents an agent using an instrument

The <e1> author </e1> of a keygen uses a <e2> disassembler </e2> to look at the raw assembly code.

- (3) PRODUCT-PRODUCER (PP): Refers to a producer causing the existence of a product

The <e1> court </e1> decided the objection by making the instalment <e2> order </e2> as sought.

- (4) CONTENT-CONTAINER (CC): Refers to an object physically stored in a defined area of space.

Flowers are nice, but don't last very long and the <e1> fruit </e1> in a fruit <e2> basket </e2> often goes bad.

- (5) ENTITY-ORIGIN (EO): Refers to an entity (either a position or material) coming or derived from an origin.

Many other <e1> dwarves </e1> also hailed from this infamous <e2> clan </e2>.

- (6) ENTITY-DESTINATION (ED): Refers to an entity moving towards a particular destination.

This book has transported <e1> readers </e1> into <e2> ancient times </e2>.

- (7) COMPONENT-WHOLE (CW): Refers to an object of a larger whole.

The <e1> owl </e1> held the mouse in its <e2> claw </e2>.

- (8) MEMBER-COLLECTION (MC): Refers to a member who forms nonfunctional part of a collection.

In the same way, a <e1> society </e1> is built up of many <e2> individuals </e2>.

- (9) MESSAGE-TOPIC (MT): Refers to a message either written or spoken corresponds to a specific topic.

The <e1> flyer </e1> advertised a two-week <e2> workshop </e2> being conducted at the university this summer.

Subtasks: For a given sentence with two tagged entities ($e1$ and $e2$, where the ordering of these tags occurs in the order of words in the sentence) such as *The <e1> burst </e1> has been caused by water hammer <e2> pressure </e2>*. the participating system should:

- (1) Predict the relationship between the entities (select one class out of ten):

Relation (burst, pressure) = CAUSE-EFFECT

- (2) Predict the direction of the relation:

(e2: pressure) caused the (e1: burst) therefore direction is from pressure to burst
 $e2 \rightarrow e1$

Similar to DDI, the second task to predict the direction of the relationship was omitted from experimenting in this thesis as the main focus was on relation extraction task and not particulars of the task related to the data set. Therefore, the classification task on Sem10 was treated a simple 10 ten way classification.

Dataset: The SemEval10 dataset consists of 10,717 annotated examples where 8000 were treated as training samples and the remaining 2717 were treated as test samples. Every data sample is a single sentence containing only a pair of entities between which the relationship and direction should be identified. Even though most of the tagged entities span over a single word, they are not restricted to it. There are entities which spans over a base noun phrase (NP)¹ whose head is a common noun as shown in Ex 5. Table 1.2 shows some basic statistics on the Sem10 data set.

Ex 5: "Feminist science fiction tends to deal with women's roles in society."

Table 1.2: Basic statistics on Sem10 data set

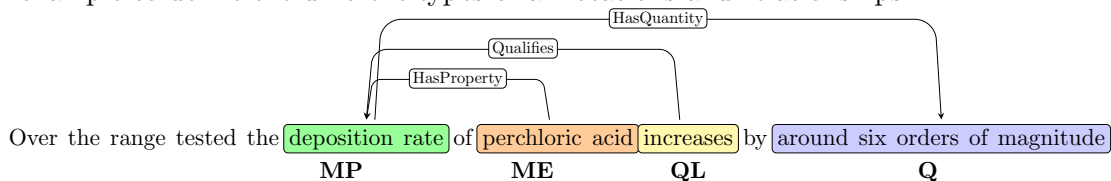
	Training	Testing
Total number of sentences	8000	2717
Total number of single word entities	15226	5190
Total number of multiple word entities	774	244

¹A base noun phrase (NP) consists of a noun and its premodifiers, such as nouns, adjectives, and determiners

1.3 MeasEval: Counts and Measurements (MeasEval)

Objective of the task: The MeasEval data set is made up of excerpts from scientific articles from Elsevier Labs OA-STM-Corpus² which is a collection of scientific, technical, and medical articles. The excerpts provided as a dataset varied in multiple disciplines of science. Counts and measurements are an important part of scientific discourse. Even though it is relatively easy to identify counts and measurements, detection of them alone is not useful unless one knows what it measures and other additional properties relevant to it. Since there is no one standard way in writing scientific documents, the ambiguous and inconsistent nature of scientific documents poses a challenge for extracting relevant information as they vary widely in terms of location with regard to measurements and well in type of information. This task was introduced in SemEval-2021 task 8 (Harper et al., 2021) with the objective of extracting counts and measurements together with other relevant information.

Subtasks: The MeasEval consists of 5 (non independent) subtasks covering entity detection, span detection, and relation extraction across multiple sentences³. Since the main objective of the task is to identify quantities (counts or measurements), related entities and additional properties of entities, the term **Annotation Set** is used in this thesis to define all related entities and relations associated with a particular quantity. Individual entities within an annotation set will be identified as **Annotation Type**. An annotation set can consist of four types of annotations and three relation types. The following sentence is used as an example to define the different types of annotations and relationships.



The four annotation types are:

- (1) QUANTITY (**Q**): Is the starting point for an annotation. It is either a *Count* consisting of a value or a *Measurement* consisting of a value and usually a unit. A Quantity can additionally include optional modifiers like ranges or approximation. (*e.g. around six orders of magnitude* is the span of the Q with *orders of magnitude* as Unit(U) and *IsApproximate* as Modifier(MOD) to indicate that the Q is a approximation). In very rare cases a Quantity can exist standalone if no other entities can be detected in the

²<https://github.com/elsevierlabs/OA-STM-Corpus>

³<https://competitions.codalab.org/competitions/25770>

document.

- (2) **MEASUREDENTITY (ME)**: Refers to an entity being measured by Q (*e.g. perchloric acid*). It is a required span of text that has Q as a direct value or indirectly through MeasuredProperty.
- (3) **MEASUREPROPERTY (MP)**: An optional span associated with both ME and QT. This refers to a span of text which resembles some property of ME, the Q is referring to (*e.g. deposition rate*).
- (4) **QUALIFIER (QL)**: Optional qualifier span describes the special circumstances which affect a particular measurement. It can be relate to any ME, MP or Q. In the given example *increases* is a QL of MP.

Annotation types are connected through 3 types of relationships; they are:

- (1) **HASQUANTITY (HQ)**: Is a relationship type which relates a MP to a Q or ME to a Q if there exists no MP in the annotation set (*e.g. deposition rate(MP)- HQ -around six orders of magnitude(Q)*).
- (2) **HASPROPERTY (HP)**: Is a relationship type relating a ME to a MP (*e.g perchloric acid(ME)- HP -deposition rate(MP)*)
- (3) **QUALIFIES (QLS)**: Is a relationship type relating a QL to either a Q or ME or MP (*e.g. increases(QL)- QLS -deposition rate(MP)*).

As mentioned above, a Q can have additional properties such as units (U) and modifiers (MOD). Units (U) denote the metric used to measure the quantity and contained within the Q span in the gold annotation. Modifiers are the type of class into which the quantities can be grouped. The following is a list of the 9 types of modifiers defined in the task with examples:

- (1) **ISAPPROXIMATE**: *The polar forcing via ion drag generates strong westward (sub-corotating) winds at a peak velocity of around 1300 ms-1 near 82° latitude.*
- (2) **ISCOUNT**: *This detection was based on four transits observed with the Space Telescope Imaging Spectrograph (STIS) onboard the Hubble Space Telescope (HST).*
- (3) **ISRANGE**: *Concentrations of salts in Mars soil assuming deposition during the Amazonian, a soiling density of 1 g cm-3, and mixing in a range of 1.5 - 2.6 m depth.*
- (4) **ISLIST**: *Effect of bed inventory on increase of solid minor element concentrations for bed inventories 4.5 kg, 6 kg and 13 kg CaCO3.*

- (5) ISMEAN: *In our solstice simulation (R19) we find exospheric temperatures averaged from $74^{\circ}S$ to $90^{\circ}S$ (the summer polar region) of 490 K.*
- (6) ISMEDIAN: *If we use a median Os abundance in seawater of 10 ppq we can evaluate the approximate Os contribution from the Caribbean LIP to the global ocean using a progressive mixing model.*
- (7) ISMEANHASD: *The external diatomite standard ($1.26 \pm 0.2\%$, Reynolds et al., 2007) yielded a mean and 2SD of $1.23 \pm 0.25\%$ ($n=104$).*
- (8) ISMEANHASTOLERANCE: *They found that the average variability during one solar rotation was found to be $9 \pm 6\%$.*
- (9) ISRANGEHASTOLERANCE: *Even higher temperatures in Saturn’s auroral oval was of $(563-624) \pm 30 K$ were derived from Cassini VIMS observations by Stallard et al.*
- (10) HASTOLERANCE: *Recent sanidine $40Ar/39Ar$ and zircon $206Pb/238U$ geochronology integrated with astrochronology constrain the CTB at $93.90 \pm 0.15 Ma$.*

Except for quantities falling into the ISCOUNT category, the rest may or may not have units. If a unit exists in the text, then it should be indicated along with the modifiers.

The complexity of the MeasEval data set, for this thesis is reduced to only relation extraction. More specifically, I only focus on the detection and classification of annotation types Q, ME and MP⁴ and the relationship between them.

Dataset: In total, the MeasEval data set consists of 428 documents (scientific excerpts), of which 313 documents are treated as training data and 135 documents as test sets. The number of sentences within these documents range from 1⁵ to 50⁶ sentences. Not all sentences within the documents contain gold annotations. Resulting in 298 documents with gold annotations for training. Table 1.3 shows basic statistics of the corpus. As shown in the table, the dispersion of annotation sets along with very few samples to train a system makes the task highly complex compared to others.

In summary, this thesis focuses on the Relation Extraction (RE) by using three different datasets, that vary in domain and complexity. RE can be defined as extracting relationship between entities of interest based on the language cues found in the text. Since dependency parse trees are capable of capturing grammatical structure of sentences, this thesis focuses on extracting and representing linguistic features based on the dependency parse

⁴QL is ignored as they do not have a proper definition as to what pertain as QL

⁵doc.id: S0378383912000130-1096

⁶doc.id: S0022000014000026-18167

Table 1.3: Basic statistics on MeasEval data set

	Training	Testing
Total number of sentences	1733	756
Total number of sentences with annotation sets	294	130
Total number of Annotation sets	866	369

tree and encoding them with varying encoding schema including deep transformer-based architectures to evaluate whether these features are suitable to perform the RE task, and whether the deep learning models benefit from these features when combined to perform the RE task. Since the objective is to evaluate different encoding schema across all data sets, a generic approach of selecting features and parameters of models was followed when designing different experiments. Due to the generic approach followed, this thesis does not focus on achieving SOTA performance on any data set. Therefore, all subtasks associated with each data sets (except the relation extraction task) is omitted from experimentation.

This thesis is structured to provide background information on the neural network models along with description of linguistic features used and different representation techniques used in Chapter 2. The following chapter focuses on the importance and capability of dependency parse trees on RE task, with a Graph Neural Network (GNN) based approach to encode it. Moreover, Chapter 3 also contains information about the data processing and extraction of features on each dataset. Chapter 4, discusses the different experiments conducted with different encoding schemas along with observation and analysis. The final chapter, Chapter 5, concludes the thesis, with future work that could be carried out based on this study.

Chapter 2

Background

2.1 Relationship Extraction (RE)

RE is one of the steps in Information Extraction (IE). [Culotta, McCallum, and Betz \(2006\)](#) defined it as:

“The task of discovering semantic connections between entities. In text, this usually amounts to examining pairs of entities in a document and determining (from local language cues) whether a relation exists between them”

Formally, a relation can be represented as $r = (e_1, e_2 \dots e_n)$ where r is a relation and e_i is an entity in a document D . Most relation extraction tasks focus on extracting binary relations such as $r(e_1, e_2)$ (e.g. *parent(mother, son)*) but can also extend to higher-order relations ($r(e_1, e_2, e_3)$) ([McDonald et al., 2005](#)). However, this thesis will only focus on the extraction of binary relations.

A binary relation is defined as a relation that exists between 2 entities. That being said, we then raise the question as to what an entity is. From a language point of view, the definition of an entity (and relations) varies depending on the genre of the text and the purpose of the task. Generally, a **Entity** simply refers to a word or a sequence of words that refers to a proper name. A proper name can refer to a person (e.g. John), place (e.g. Germany), organization (e.g. WHO), facility (e.g. YUL airport), etc. In NLP, the process of detecting proper names and categorizing them is known as Named Entity Recognition (NER). However, specialized applications could involve other types of entity such as proteins and genes in the biomedical domain ([Segura-Bedmar et al. \(2011\)](#); [Segura-Bedmar et al. \(2013\)](#)) and commercial products or works of art in the e-Commerce sector, etc. Even though the notion of an entity refers to the identification of proper names, it can be extended to things which cannot be explicitly seen as entities but have practical

importance and characteristics which signal them as entities, such as *temporal expressions* (Gast, Bierkandt, Druskat, and Rzymyski (2016); Bethard et al. (2016)) which includes dates, time and named events, and *numerical expressions* such as measurements (Harper et al., 2021), counts, prices, etc.

Once the entities are identified, the relations between the entities are extracted using the relation extraction technique. Relation extraction has been an active topic in the research field for a long time. To explore effective ways of extracting relationships from text, new data sets are released from time to time in the form of competition (*Multi-way classification of Semantic Relations* (Hendrickx et al., 2010); *New York Times Corpus* (Riedel, Yao, & McCallum, 2010); *Drug-drug interaction corpus* (Segura-Bedmar et al., 2013); *WikiData for Sentential Relation Extraction* (Sorokin & Gurevych, 2017); *Tacred* (Y. Zhang, Zhong, Chen, Angeli, & Manning, 2017); *Capturing discriminative attributes* (Krebs, Lenci, & Paperno, 2018); *Few-shot relation classification (FewRel)* (Han et al., 2018); *FewRel2* (Gao et al., 2019); *MeasEval* (Harper et al., 2021)). Early work on RE focused on extracting relations from single sentences with entities explicitly annotated by task organizers. However, with the use of deep learning approaches, the intensity of the tasks continues to grow to address various practical requirements, such as detecting entities as the first step before detecting a relationship, as in MeasEval (Harper et al., 2021); or spanning relation detection across multiple sentences, as in DocRED (Yao et al., 2019) and BioCreative V CDR (J. Li et al., 2016); or performing zero-shot learning to extract new relation types that are only specified during the test time, which can be used to construct knowledge graphs and does not require the hard labor of human annotation (Levy, Seo, Choi, & Zettlemoyer, 2017).

Different machine learning techniques were used to perform the RE task. Jurafsky and Martin (Jurafsky & Martin, 2009) broadly cluster these techniques into two main groups:

- supervised methods.
- semi-supervised methods, or self-supervised methods.

This thesis addresses the RE task as a supervised task. Supervised methods treat RE as a classification problem where text is provided with relations explicitly annotated by human analysts. In general, a classification task would have positive and negative training examples. Therefore, given a sentence S with entities e_1 and e_2 , the output of the classifier is given as:

$$f_R(T(S_{e_1, e_2})) = \begin{cases} +1 & \text{If } e_1 \text{ and } e_2 \text{ are related by } R \\ -1 & \text{Otherwise} \end{cases} \quad (2.1)$$

Here, $f_R(\cdot)$ refers to any discriminative classifier function and $T(S_{e_1, e_2})$ refers to the transformation of a sentence into a vector representation of extracted features. The following sections discuss the background of different classification models and linguistic features that are used as input to these models along with the feature representation techniques used in this thesis.

2.2 Classification Models

Classification refers to a predictive modeling problem in which a class label is predicted for a given input example. To achieve this, in machine learning (ML), a model is trained using the training data set to best map the input data to specific class labels. From rule-based approaches to deep learning techniques, the evolution of ML techniques has greatly influenced the field of NLP in terms of the complexity of the tasks addressed and the performance achieved.

2.2.1 Feed-Forward Neural Network

Also known as Multilayer Perceptron (MLP), it is one of the fundamental techniques, consisting of one or many layers of fully or partially connected neural nodes, allowing the approximation of complex functions (Pal & Mitra, 1992). Each layer in a neural network is made up of one or more nodes that receive an input in the form of a linear combination of the output of the previous layer. These inputs are then passed through a non-linear transformation (see Table 2.1) function or activation function that result in the output of the current layer.

Table 2.1: Activation functions

Name	Activation functions
Softmax	$softmax(x)_i = \frac{exp(x_i)}{\sum_i exp(x_i)}$
Sigmoid	$sigmoid(x) = \frac{1}{1+exp^{-x}}$
Hyperbolic tangent	$tanh(x) = \frac{exp^x - exp^{-x}}{exp^x + exp^{-x}}$
Rectified Linear Units	$ReLU(x) = max(x, 0)$

More formally, if x is an input vector of dimension n of a layer with m nodes and $f(\cdot)$ is a non-linear function, then the m dimensional output y of the layer is computed as:

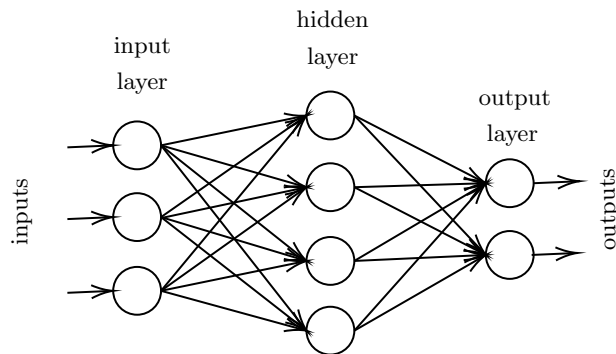


Figure 2.1: Illustration of a fully connected feed-forward neural network

$$y = f(Wx) \tag{2.2}$$

where W is a weight matrix $n \times m$, learned during the backpropagation phase (Rumelhart, Hinton, & Williams, 1986) while training the neural network.

2.2.2 Support Vector Machines (SVM)

Being effective in high-dimensional spaces and versatile in using different kernel functions made SVM a strong contender in RE shared tasks (Lai, Leung, and Leung (2018); Chowdhury and Lavelli (2013); Speer and Lowry-Duda (2018); Thomas, Neves, Rocktäschel, and Leser (2013)). A binary SVM (Cortes & Vapnik, 1995) is based on the notion of identifying a hyperplane that best divides a data set into two. As shown in Figure 2.2, the SVM algorithms identify critical data points named *support vectors*, which determine the position of the dividing hyperplane. A *hyperplane* can be defined as a $n - 1$ subspace that linearly separates a space of n . Therefore, for a 2 dimensional space, the hyperplane would be a 1 dimensional dot, whereas for a three-dimensional space, the hyperplane would be a two-dimensional line, etc. Intuitively, the further the data points are from the hyperplane, the more confident they are that they have been classified correctly. Therefore, the SVM algorithm tries to find an optimal hyperplane by maximizing the margin between the support vectors, where *margin* refers to the distance between the support vectors (that is, the closest data point) from either class to the hyperplane.

Finding a hyperplane that correctly classifies 100% in a real data set is impossible. Most real data sets are not linearly separable. Non-linearly separable cases are dealt with by SVM using two concepts: **Soft Margin** and **Kernel Tricks**. The soft margin tries to find a hyperplane to separate the data set with a certain amount of tolerance for incorrectly classified data points. On the other hand, the kernel trick tries to find a non-linear decision

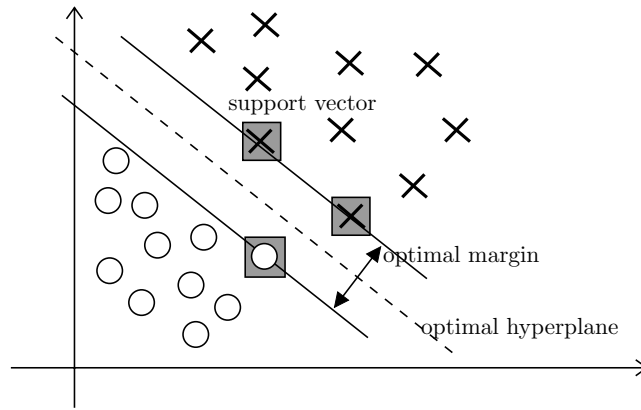


Figure 2.2: Hyperplane and support vectors in SVM

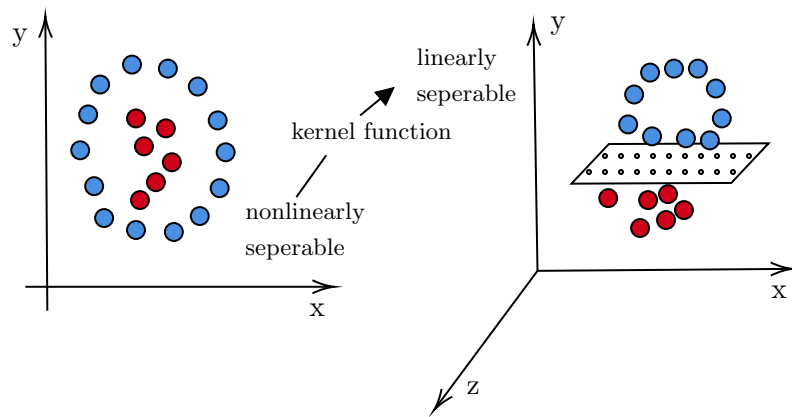


Figure 2.3: Transformation of features to higher dimension with kernel trick.

boundary by applying transformation functions to existing features, casting the data points to a higher dimension (see Figure 2.3) to find a hyperplane to divide the data set.

2.2.3 Convolutional Neural Networks (CNN)

CNN employs a mathematical operation called *convolution*, which is a linear operation that involves multiplication of weights with input to produce an output. This operation is performed in a way that can be simply thought of as a sliding window function on a matrix. Introduced in the field of computer vision by (LeCun, Haffner, Bottou, & Bengio, 1999), CNN gained popularity (Krizhevsky, Sutskever, and Hinton (2012); Oquab, Bottou, Laptev, and Sivic (2014)) due to its ability to capture localized information that was shared between neighbors in the input data (that is, features such as edges and shadows are identified in images based on the surrounding pixel). Inspired by its greater ability to capture localized characteristics, Kim (2014) introduced CNN to NLP to capture spatial information between words, resulting in improved performance in a variety of NLP tasks (Nguyen and Grishman

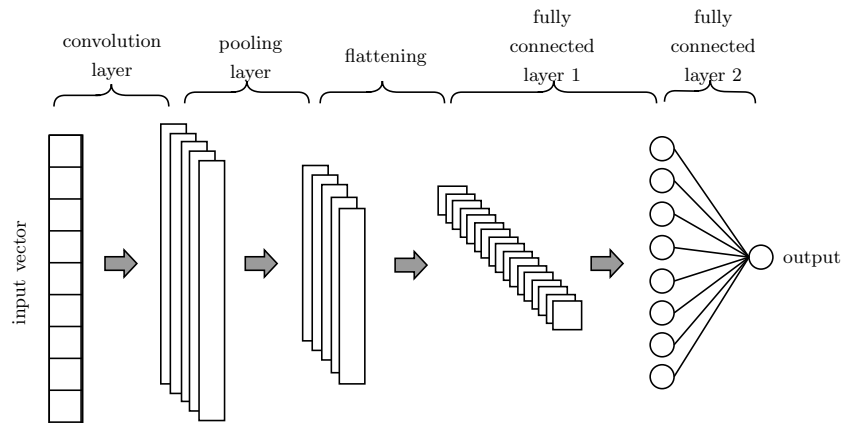


Figure 2.4: 1D Convolution Neural Network

(2015); Kim (2014)) and RE tasks (Shen and Huang (2016); Wang, Cao, de Melo, and Liu (2016); Zeng, Liu, Lai, Zhou, and Zhao (2014)).

Convolutions vary in 1D, 2D, or 3D depending on how the convolution functions are applied to the input data. This thesis uses a 1D convolution as part of the experimental setup, where the convolution window moves only in a single direction. A CNN architecture is made up of one or more layers of convolution layers, pooling layers, and finishing it off with a fully connected feed-forward network for the final classification.

Convolution in the convolution layer is performed by a *kernel* (weight matrix) that operates in a region of input data to return a single value sum as the output of an element-wise dot product. This operation is performed on all input data by sliding the kernel window in a single direction, where the sliding size is determined by the value *stride*. A combination of kernel windows are used in the input data to obtain different abstractions of features.

Pooling performs a scalar transformation on each local region of input data; however, unlike the kernel function, pooling either computes the average of values (average pooling) in the region it operates or selects the highest value in the region, discarding the rest (max pooling) or the opposite in min pooling. The idea of pooling is used to extract only the necessary information while reducing the effects of noise on the data. Additionally, the pooling greatly reduces the computational cost of the CNN architecture as it generally reduces the dimension size of the input data. After multiple layers of interchanged convolution and pooling layers, the final layer of the CNN architecture is the fully connected feedforward network, which performs the classification task.

Although CNNs were able to capture localized linguistic characteristics and improve performance, they were not suitable for dealing with inputs of varying length, such as sentences. Therefore, to accept input of varying length and preserve word ordering in

sentences, Recurrent Neural Networks (RNN) (Elman, 1990) were introduced, followed by LSTM (Long Short-Term Memory) networks (Hochreiter & Schmidhuber, 1997) to address problems of vanishing gradients, which prevent RNN networks from maintaining long-range dependency in sentences. Although RNN and LSTM are used in many RE tasks (Socher, Huval, Manning, and Ng (2012); Lin, Shen, Liu, Luan, and Sun (2016); Cai, Zhang, and Wang (2016)), the details of these architectures will not be discussed, as they are not experimentally investigated in this thesis.

2.2.4 Transformers

The transformer Vaswani et al. (2017) is a sequence-to-sequence architecture. A sequence-to-sequence architecture transforms a given sequence of source inputs into another sequence of target output, similar to machine translation (MT). This transformation is carried out using a **encoder** and **decoder** mechanism. The encoder takes an input sequence of arbitrary length and maps it into a higher-dimensional space (i.e. a fixed-length vector known as the context vector), which is then fed into the decoder to transform it into an output sequence. An LSTM/RNN model-based sequence-to-sequence architecture performs this translation by providing the last hidden state of the encoder as the context vector, containing the summary of the input sequence, as input to the decoder for translation. Here, the context vector is expected to remember all the necessary information in the source sentence, in a fixed size vector. This becomes a bottleneck for longer sentences, as the model forgets some information from the earlier part of the input sentences, producing bad context vectors, resulting in poor performance.

As a remedy for this bottleneck of the fixed-size context vector, Bahdanau et al. (2015) proposed a mechanism, namely **attention**, to take into account all input words in the context vector while providing relative importance to each word in the input. This concept is widely used in the RE task (Lin et al. (2016), Shen and Huang (2016), Wang et al. (2016)) to obtain a context vector based on the relative importance of the annotated entities. This relative importance process is similar to the cognitive process of selectively concentrating on one or a few things while ignoring others. Thus, whenever a decoder generates a sentence given an input, it searches for a set of positions in the encoder hidden states where the most relevant information is available. Extracted from the paper Bahdanau et al. (2015), Figure 2.5 shows the intensity of the attention weights (brightness) with relevance to the source to target word in the English-French machine translation task. These attention weights are learned by the network, while it adjusts itself according to the output element in the target sequence generated by the decoder. Since the context vector has access to all

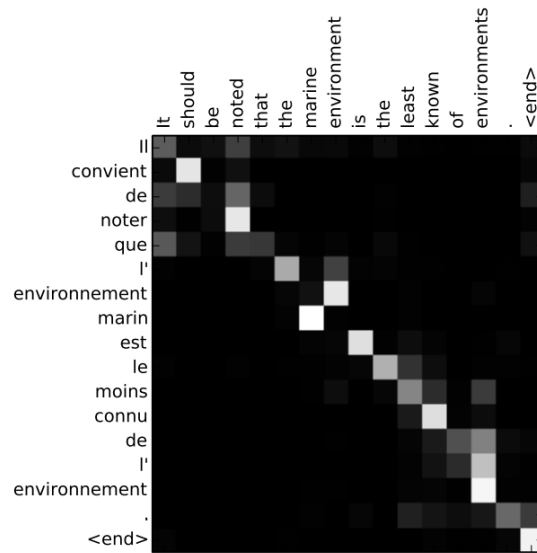


Figure 2.5: Attention weights associated with source words (English) to target words (French) in English-French machine translation task (Bahdanau et al., 2015)

elements of the input sentence, long dependencies could be retained regardless of the length of the source sentence.

While attention is focused on relating words from one sequence to another, **self-attention** (Cheng et al., 2016) focuses on relating different positions within the same sequence to compute a representation of the sequence. Extracted from the paper Cheng et al. (2016), Figure 2.6 illustrates the different memories or words triggered when processing words in red. The different shades of blue represent the degree of memory activation. Focusing on the same concept (self-attention), Vaswani et al. (2017) came up with an alternative way to calculate self-attention than Cheng et al. (2016). Furthermore, Vaswani et al. (2017) introduced the **multi-head attention** mechanism to project each input in different *representation subspaces*, allowing the model to attend parts of the sequence differently.

Using the multi-head attention mechanism Vaswani et al. (2017) introduced the transformer architecture consisting of two parts, the encoder and the decoder. The proposed architecture consists of 6 **Encoders** stacked on top of another, where each stack contains 2 sub-layers, as shown in Figure 2.7. The first is the multi-head attention mechanism consisting of eight independently trainable self-attention heads, and the second is a position-wise fully connected feed-forward network. Both sub-layers are followed by a residual connection and a normalization layer, before the values propagate to the next layer. The first encoder in the stack accepts the input as word embeddings summed with the positional encoding calculated as shown in Equation 2.3:

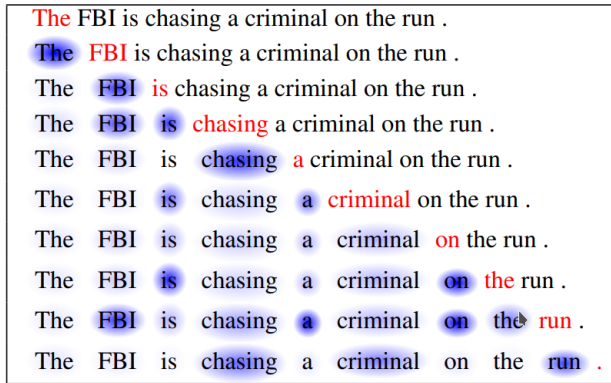


Figure 2.6: Different memory activation indicating the influence of memory words (blue) on the target word (red). Intensity of blue indicates the degree of memory activation. (Cheng et al., 2016)

$$PE(t, i) = \begin{cases} \sin(t/10000^{2k/d_{model}}), & \text{if } i = 2k \\ \cos(t/10000^{2k/d_{model}}), & \text{if } i = 2k + 1 \end{cases} \quad (2.3)$$

where $PE(t, i)$ denotes the positional encoding vector of the i^{th} index of the embedding vector of the token in position t . Integration of positional encoding is performed to incorporate position information into the token embeddings, as transformer encoders process the input in parallel. This contrasts with RNN or LSTM based encoders, where input is processed sequentially. The remaining encoders on the stack accept the output of the previous encoder as input.

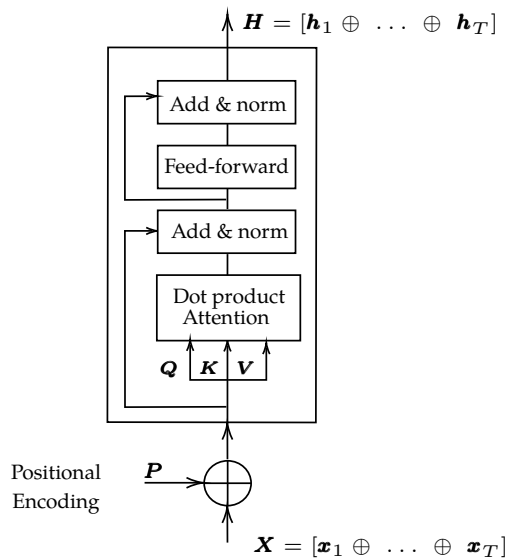


Figure 2.7: Illustration of Transformer Encoder architecture (Vaswani et al., 2017)

Similarly to the number of encoders, the proposed transformer architecture consists of **6 Decoders** stacked on top of another, consisting of two sublayers as in the encoder and an additional sublayer to apply masked multihead attention to the output embedding. This masked multi-head attention is used to prevent decoder from being aware of the future tokens to be predicted, but allowing already predicted tokens as well as the current token to be input to the decoder. The second multi-head attention layer in the decoder accepts the encoder output as input, as well as the output of the masked multi-head attention layer, allowing it to decide which encoder input to focus on. The output of each decoder is passed along the stack of decoders, and the final output of the decoder is passed through a linear classifier and softmax functions to output the probabilities of the next possible token in the sequence.

2.3 Linguistic features

A **feature** is an individual measurable property or characteristic of a phenomenon.

(Bishop, 2006)

As with any NLP task, a text document goes through a series of pre-processing steps to clean the data and extract the necessary features to perform the task. How we perform cleaning or extraction and what features are extracted have a considerable influence on the overall performance of the system. Since textual preprocessing is vital and many researches have been conducted throughout the years, currently there exist many different packages for facilitating this task using different approaches:

NLTK (Natural Language Take Kit) (Bird, Klein, & Loper, 2009) is a suite of Python libraries and programs specially used for symbolic and statistical processing of languages, widely used in the field of teaching and research;

Stanford CoreNLP (Manning et al., 2014) provides a set of language processing tools written in JAVA, supporting multiple languages like Arabic, French, Chinese, etc. in addition to English;

spaCy¹ is a free open-source library based on the neural net for language processing. Unlike other packages, spaCy focuses more on software for production use, facilitating the integration of different popular machine learning libraries such as PyTorch²,

¹<https://spacy.io/models>

²<https://pytorch.org/>

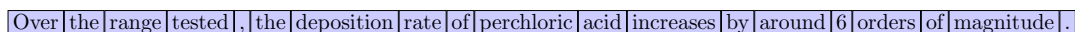


Figure 2.8: Unicode tokenization

TensorFlow³ and others within its pipeline;

GATE (Cunningham, Tablan, Roberts, & Bontcheva, 2013) (General Architecture for Text Engineering) is a suite of JAVA-based frameworks that facilitate language processing and provide a user interface (UI) for visual analysis of annotations, which contributes heavily to the process of finding language-related patterns.

Regardless of the package, a typical NLP preprocessing pipeline includes sentence splitting, tokenization, sometimes followed by task-relevant gazetteer annotation, possibly Named-Entity-Recognition(NER), part-of-speech (POS) tagging, and dependency parsing. The following sections provide brief descriptions of some important steps in the preprocessing pipeline with sample annotations based on Ex 6.

Ex 6: “Over the range tested, the deposition rate of perchloric acid increases by around 6 orders of magnitude.”

2.3.1 Tokenization

Tokenization is the process of splitting text into basic units called tokens which are composed of a set of characters, such as words or terms. Tokenization is a crucial step in the preprocessing pipeline, since the meaning of the text is interpreted by analyzing the tokens existing in a text. Different tokenization techniques are used to satisfy different requirements. ANNIE’s (Cunningham, Maynard, Bontcheva, & Tablan, 2002) Unicode tokenizer (used in GATE) splits text into very simple tokens such as numbers, punctuation marks, symbols, and words of different types such as uppercase and lowercase, preserving the surface-level lexical information to be capitalized by the other modules in the preprocessing pipeline. Figure 2.8 illustrates the output of the GATE tokenizer for Ex 6.

In contrast, pre-trained language models such as BERT (Devlin et al., 2019) and XLNet (Z. Yang et al., 2019) use *Subword Tokenization*, also known as *Wordpiece Tokenization* (Y. Wu et al., 2016), which preserves commonly used words as is and decomposes rare words into meaningful sub-words. Therefore, Ex 6 will be divided into 21 tokens, where the rare word “perchloric” is broken down into 3 tokens such as “perch”, “##lor”, and “##ic” as shown in Figure 2.9. This enables BERT-like modules to generate somewhat meaningful representation for unknown words by combining word pieces that are shared

³<https://www.tensorflow.org/>

Over	the	range	tested	,	the	deposition	rate	of	perch	##lor	##ic	acid	increases	by	around	6	orders	of	magnitude	.
------	-----	-------	--------	---	-----	------------	------	----	-------	-------	------	------	-----------	----	--------	---	--------	----	-----------	---

Figure 2.9: Wordpiece tokenization

with other similar tokens. In addition to the Unicode tokenizer or the WordPiece tokenizer, there are other techniques such as *character-based tokenizer*, *Regex tokenization*, etc. which will not be discussed in detail as they are not used in this thesis. Since tokenization usually appears as the primary step in the preprocessing pipeline, errors in this stage propagate through other modules, producing erroneous features. Therefore, choosing an appropriate tokenization technique for the task and fixing errors in this stage is crucial.

2.3.2 Part-of-Speech (POS) tagging

POS tagging is the process of assigning each token in the sentence a tag corresponding to a particular part of the speech, based on both its definition and context. Or in other words, it can be defined as indicating for each word whether it is a noun, verb, adjective, adverb, preposition, etc. Penn Treebank (Marcus, Marcinkiewicz, & Santorini, 1993) is a commonly used POS tag set consisting of 36 POS tags and punctuation marks. Figure 2.10 shows an example of POS tagging for the sentence in Ex 6 which is tokenized based on the Unicode tokenizer of the ANNIE module (Cunningham et al., 2002) in the GATE pipeline.

Over	the	range	tested	,	the	deposition	rate	of	perchloric	acid	increases	by	around	6	orders	of	magnitude	.
IN	DT	NN	VBN	,	DT	NN	NN	IN	JJ	NN	NNS	IN	RB	CD	NNS	IN	NN	.

Figure 2.10: POS tagging of Ex 6 from Stanford Parser (Klein & Manning, 2003)

2.3.3 Dependency Parsing

Also known as *typed dependency parsing*, describes the syntactic structure of a sentence in terms of binary grammatical relations between words. Relations between the words are indicated with directed, labeled arcs from heads (governors) to dependents. Since the relation labels are from a fixed grammatical inventory, the formalism of dependency relations is known as typed dependency parsing. More formally, a dependency parse tree is a graph $G = (V, E)$ where V is a set of vertices that contain words in a sentence, and the edge E is the dependency relationship that connects two words. In addition to the tree-like structure, a dependency parse tree must satisfy 3 other conditions: 1) There must be a single *root* node with no incoming edges. 2) For each node v in V , there must be a path from root R to v . 3) Each node except the root must have only a single incoming edge (that is, a

child node (dependent) can only have a single parent node (governor)). Figure 2.11 shows a typed dependency parse tree of Ex 6 using the dependency parse grammar formalism of the Stanford dependency parser (de Marneffe et al., 2006) and Figure 2.12 illustrates the tree-like perspective that could be derived from the dependency parse tree.

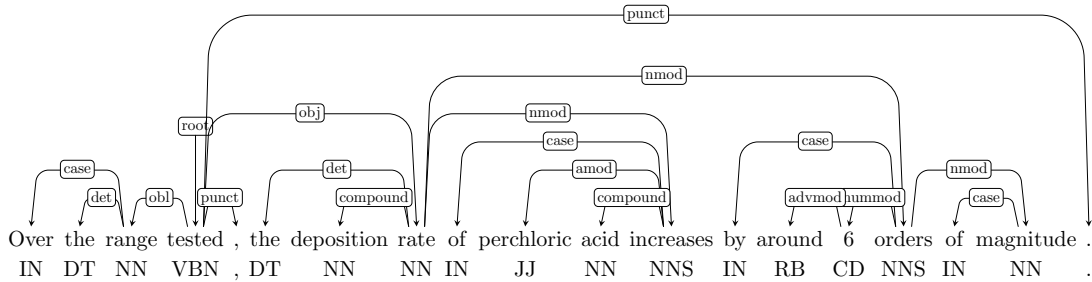


Figure 2.11: A typed dependency parse from the Stanford Parser (de Marneffe et al., 2006)

2.4 Feature Representations

Features used to represent data can be either numerical (e.g. age, height, weight, volume, etc.) or categorical (e.g. name, gender, sex, token strings, POS tags, etc.). In principle, a neural network requires input as a vector of real numbers. Numerical features extracted from the data can be represented as it is since they are numbers and neural networks can handle them. On the contrary, categorical features must be transformed into a numerical vector before feeding to a neural network. There are multiple techniques available for the transformation of categorical data into a numerical vector representation. The following is a list of representation techniques that are used in this thesis to represent the linguistic features.

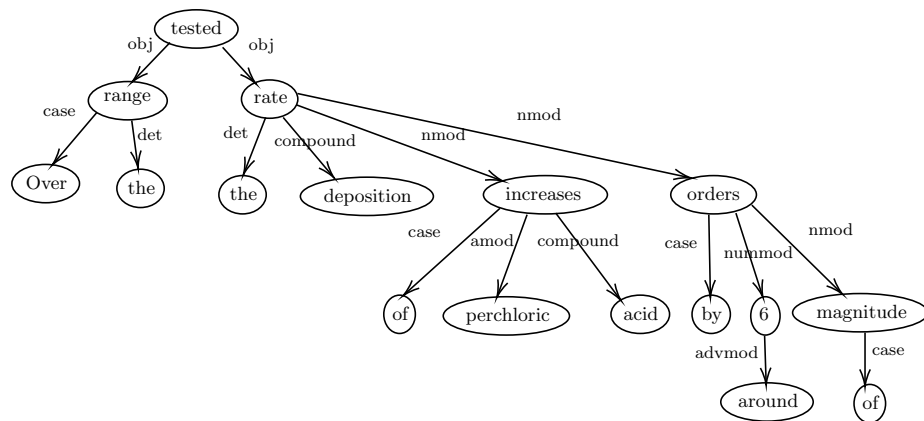


Figure 2.12: Tree-view of dependency parsed sentence shown in Figure 2.11

2.4.1 One-hot-encoding

It is a very simple and straightforward technique that requires very little computational effort and time to construct. Formally, one-hot encoding is defined as x being any discrete random categorical variable with n distinct values x_1, x_2, \dots, x_n . The one-hot vector v of a value x_i will be of size n with $n - 1$ components of v filled with 0 except the i^{th} component, which will have a value 1. For example, given a set C corresponding to a set of blue, green, and red colors $C = \{b, g, r\}$, a random variable x taking values from C will have one-hot encoding as: $(1,0,0)$, $(0,1,0)$, $(0,0,1)$ for b, g, r , respectively. Regardless of its simplicity, the disadvantage of this technique is that the dimension of the vector increases with the size of the unique values (*vocabulary size*) a feature contains, and regardless of the size of the vector, a vector will contain only 1 bit of information. This leads to sparsity, resulting in higher consumption of memory and issues related to higher dimension (curse of dimensionality).

The *curse of dimensionality* is an expression coined by Richard Ernest Bellman (Bellman, 1961) describing the phenomenon that arises when analyzing/organizing data in a high-dimensional space. Among the many potential issues, containing more features than observation leads machine learning models to overfit due to a lack of data to generalize based on the observed features. Especially for algorithms based on distance measuring (e.g. K-nearest neighbors), the more the dimension, the more the distance between points, leading for data points to appear equidistant from all others, preventing meaningful clusters to be formed.

2.4.2 Label encoding

Is a technique that provides a unique integer label to each categorical value of a given feature. Similarly to the color example above (assuming the colors are sorted alphabetically), label encoding would provide label 1 for blue, 2 for green, and 3 for red. Even though this is one way of encoding and this works for some variables, such as ordinal values (*first, second, third*), this does not suit all categorical variables (e.g. colors). Why? Real numbers have a natural ordering relationship, which in return imposes an artificial ordering among the categorical variables, resulting in unpredictable results. Also, in neural networks, during the optimization step with stochastic gradient descent, the value of a feature with a larger encoded label will contribute more during weight updates. Regardless of its downside, label encoding is still a useful technique utilized to encode class labels in a multiclass classification task, as well as in decision tree-based algorithms, which works well with categorical data.

2.4.3 Multi-hot encoding

Also known as binary encoding, it is a compromise between label encoding and one-hot encoding. In this approach, the categorical values are encoded first by label to obtain a unique value and then convert the numerical labels into a binary vector of size $\lceil \log_2 n \rceil$, where n is the number of unique values in the categorical feature. Following the above example of colors, $n = 3$ therefore the size of the vector is $\lceil \log_2 3 \rceil = 2$, therefore $[0, 1]$, $[1, 0]$, $[1, 1]$ will be the representation for b, g, r respectively. Although multi-hot encoding comparatively reduces the sparsity of the feature vector, it introduces false additive relationships, e.g. $[0, 1] + [1, 0] = [1, 1]$ meaning $red + blue = green$.

2.4.4 Word Embedding

The main pitfalls of one-hot encoding discussed above are the lack of preservation of word order, word-sense information, and semantics. In other words, when we try to visualize one hot vector in a vector space, each word in the vocabulary will be a dimension on its own that has nothing to do with the rest of the dimension, resulting in all words being equally different from each other (*i.e.* “good” and “brilliant” is equally different as “house” and “dog”), which is not true. Therefore, to reflect the syntactic or semantic similarity of words, the notion of *distributed representations of words* (Hinton, McClelland, & Rumelhart, 1986) was used to create a representation of words that incorporates the dependency of its context on its dense vector representation. This dense representation is known as word embedding.

Distributed Representations refers to the representation of any single concept, distributed over many, if not all, processing units (Hinton et al., 1986).

Word embeddings (Bengio, Ducharme, Vincent, and Janvin (2003), Mikolov, Chen, Corrado, and Dean (2013)) learn word representations using a predictive neural model. Here, the model is trained to predict a target word providing a window of context/surrounding words. The hidden weights of the learned neural model correspond to the embedding of words in the vocabulary. There are many different approaches to creating word representation, and some are provided in detail below.

2.4.4.1 Word2Vec

Is a two-layered neural network proposed by Mikolov et al. (2013) to learn word representations in the form of dense word embeddings. Word2Vec aims to learn word embeddings in such a way that words sharing a similar context tend to appear closer to each other in

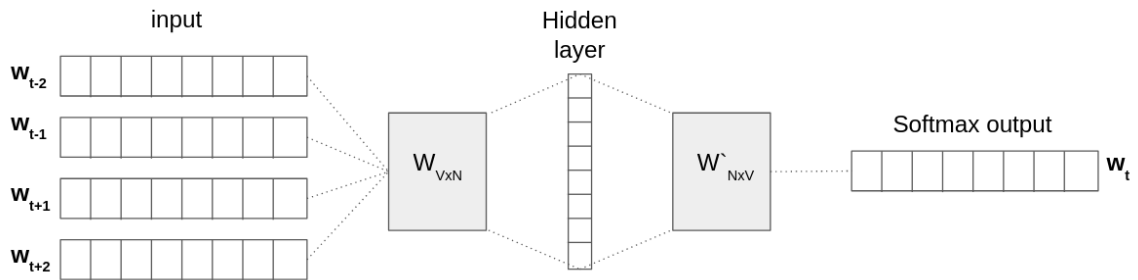


Figure 2.13: Architecture of Word2Vec Continuous Bag-of-words (CBOW) model

the vector space than words that do not often occur in the same context. Two model architectures were proposed by Mikolov et al. (2013): *continuous bag-of-words (CBOW)* and *skip-gram*. The CBOW model takes context as input and tries to predict the target word; in contrast, the skip-gram model takes the target word as input and tries to predict the context words. The context used in both approaches is formulated by a fixed size window of $(2 \times c + 1)$, where c refers to the size of the context window, and $\times 2$ is used to consider the left and right words of the target word. Given a sentence “*The more you know, the more you realize you know nothing*”, Table 2.2 shows the context-target pairs of context window size (c) two to train the neural model.

Table 2.2: Sample context-target pairs generated, with context window size 2 for sentence: “*The more you know, the more you realize you know nothing*”

Context	Target
more, you	The
the, you, know	more
the, more, know, the	you
more, you, the, more	know
you, know, more, you	the
know, the, you, realize	more
the, more, realize, you	you
more, you, you, know	realize
you, realize, know, nothing	you
realize, you, nothing	know
you, know	nothing

Given a vocabulary size V , both target word and words in context are represented with a one-hot vector of size V . Figure 2.13 shows the architecture of the CBOW model, where the number of neurons in the hidden layer N corresponds to the size of the embedding one intends to produce, and the output (i.e. the target word) is a vector of size V with

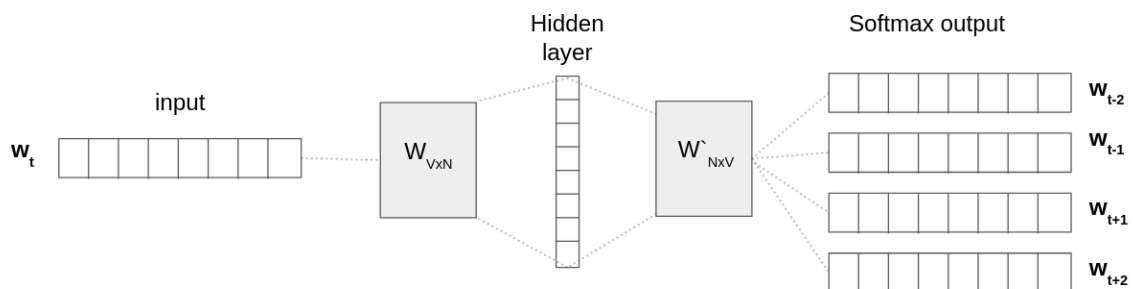


Figure 2.14: Architecture of Word2Vec skip-gram model

softmax values as elements. The softmax values indicate how probable each word in the vocabulary to be the target word given a specific context. Since the target value is one-hot encoded, the difference between the target value and the softmax values are calculated and backpropagated as error to train the model. By the end of training, the entire model is discarded except for the trained weight matrix after the hidden layer $W'_{N \times V}$, is used as a table of reference to find corresponding word embedding for all the word in the vocabulary.

Skip-gram model on the contrary can be considered as the inverse of CBOW architecture as shown in Figure 2.14, where the input is the one-hot vector of the target and the output is a probability distribution over all words in the vocabulary, for each word in the context. The training is conducted in a similar way as CBOW, where the error of each context word would be backpropagated through the model to train the weights, and the weight matrix of the hidden layer ($W'_{N \times V}$) will be preserved for obtaining the word embeddings.

2.4.5 Generalized Language Models

Even though Word2Vec learns word embedding by training the network model to predict the target words based on the context, when retrieving the word embeddings for each word there is only one representation, resulting in a context independent vector representation. In other words, given the following 2 sentences, “*I deposited my money in the bank*” and “*I saw a boy playing along the river bank*”, even though the word “*bank*” refers to two different concepts, the pretrained word embedding retrieved from the Word2vec model remains the same. To mitigate the shortcomings of this approach, approaches to compute contextualized word embeddings were proposed which computes the embedding through dynamic functions of the the context.

2.4.5.1 Embeddings for Language Models (ELMo)

To embed context dependant information in word embeddings, [Peters et al. \(2018\)](#) proposed **ELMo** (Embeddings for **L**anguage **M**odels), a deep contextualized word representation. A contextualize embedding refers to a representation that captures both the word meaning along with the information available in the context. Therefore ELMo, unlike word2vec utilizes a two layered bi-LSTM, where each layer of the bi-LSTM has two passes - forward pass and backward pass stacked together. This architecture uses character level convolution neural network to represent words of a text string into raw word vectors. These raw word vectors are used as input to the first layer of bi-LSTM. The forward pass within the bi-LSTM layer contains information about a certain word and the context (other words) that appear before the word, whereas the backward pass contains information about the word and the context that follows the word. This pair of information from the forward and backward pass forms the intermediary word vector which is then used as input to the next layer of the bi-LSTM. Final representation of ELMo is computed by obtaining the weighted sum of the raw word vectors and the two intermediate word vectors results after each layer of bi-LSTM. This enables the model to produce word embeddings based on the context surrounding, meanwhile encoding morphological information, due to the utilization of character convolutions. Even though ELMo improved in performance, however, for each specific downstream task, the model has to learn a linear combination of the hidden states of the ELMo model (i.e. the learned weights of the linear combination of the ELMo layers) to form the word embeddings resulting in training the model for every downstream task.

To mitigate training a model for every downstream task as in ELMo, **ULMFiT** ([Howard & Ruder, 2018](#)) was proposed as a solution for transfer learning, which was widely used in computer vision tasks. This philosophy introduced in ULMFiT paved for new techniques to effectively transfer knowledge learnt, but adjust to downstream task by fine tuning on the task dataset, rather than having to train a new model from scratch. Apart from ULMFiT, GPT ([Radford & Narasimhan, 2018](#)) and BERT ([Devlin et al., 2019](#)) followed the same principal of transfer learning, enabling the use of pretrained language model to be directly incorporated on end tasks. These models follow a two step approach: pretraining and fine-tuning. The following section describes both pre-training and fine-tuning with respect to BERT as it was used in this thesis.

2.4.5.2 BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers or in short **BERT** ([Devlin et al., 2019](#)) is a language model solely based on Transformer architecture (see Section 2.2.4),

more precisely a multi-layer bidirectional encoder from the transformer architecture. The introduction of BERT in late 2018, took NLP by storm, by obtaining state-of-the-art results on many NLP benchmark tasks. Even though BERT is a language model (LM), the authors of BERT rephrased the LM prediction task into a “*Masked Language Model*” (MLM) task and “*Next Sentence Prediction*” NSP tasks giving them the leverage of using birectionality of transformer architecture when producing contextualized word representation.

Statistical models, which are trained to predict the next word given a sequence are called Language Models or LMs

(Jurafsky & Martin, 2009)

As shown in Figure 2.15 the input of BERT is a representation of the sum of three embeddings:

- **Token embeddings:** are retrieved through Wordpiece tokenization (see Section 2.3, Tokenization), which allows the model to better handle unknown words by breaking down a rare word into smaller sub-word units.
- **Segmentation embeddings:** were motivated by certain downstream tasks such as Natural Language Inference (NLI) and Question and Answering (QA), where the tasks focuses on relation between two sequences. In this case the segmentation embedding is utilized to form embedding for different sequences, by having a special separation token *[SEP]* in between.
- **Positional embedding:** Unlike in sequential learning models like LSTMs and RNNs, the utilization of multiple Transformer blocks with multi-head attention mechanism, causes BERT to process text in parallel. This leads the model to not retain the word ordering. Therefore, to provide the notion of word ordering positional embedding is incorporated as part of the input embedding.

In addition to the above mentioned embeddings, a special token *[CLS]* is added at the beginning of the input. The final hidden state of this token retrieved from the BERT model, corresponds to the aggregation of sequence representation, and is used for prediction in classification based downstream tasks.

As mentioned earlier, BERT follows the philosophy of transfer learning. This is accomplished in 2 phases: pretraining and fine-tuning. In the **pretraining phase**, Bert is trained for both MLM and NSP tasks. *Masked Language Model* (MLM) task was performed by masking 15% of the tokens in the input in random and trying to predict what

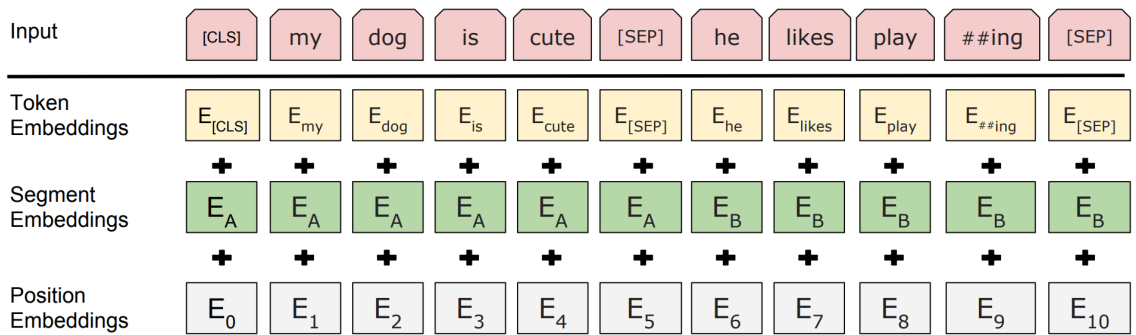


Figure 2.15: Input embeddings of BERT consisting of token embeddings, segmentation embeddings and positional embeddings. Source (Devlin et al., 2019)

the masked tokens are. The *Next Sentence Prediction* (NSP) task was performed by providing pairs of sentences to the model, and trying to predict whether the second sentence is the actual next sentence following the first one. Both MLM and NSP were performed simultaneously while training the model, therefore the weight adjustments were based on the training loss which is the sum of the mean likelihood of MLM and the mean likelihood of NSP. Since BERT was trained on BooksCorpus (800M words) and English Wikipedia (2,500M words), it provides a better generalized representations that could be utilized directly on downstream tasks. The authors of BERT released 4 different versions of BERT that could be used on different circumstances based on the requirement, BERT-base-uncased, BERT-base-cased, BERT-large-uncased and BERT-large-cased. Both BERT-base models consist of 12 transformer blocks within their architecture with approximately 110 million parameters to be trained. On the contrary BERT-large models consist of 24 transformer blocks with roughly 340 million parameters to be trained.

In the **fine-tuning** phase, a few parameters are added to the pre-trained model according to the downstream task. As shown in Figures 2.16 a and b, the final hidden representation of $[CLS]$ is used as input for any classification model, such as a feed-forward neural network, linear regression, etc. to predict the probability of classes in single sentence classification tasks or sentence pair classification related tasks. We will see how a similar construct to the above is utilized in our Relation Extraction (RE) task in Chapter 4.

For token classification tasks, such as NER detection (Figure 2.16.d), each hidden token state is used to calculate a probability distribution over a given set of c available classes using a neural network, to predict which class each token belongs to.

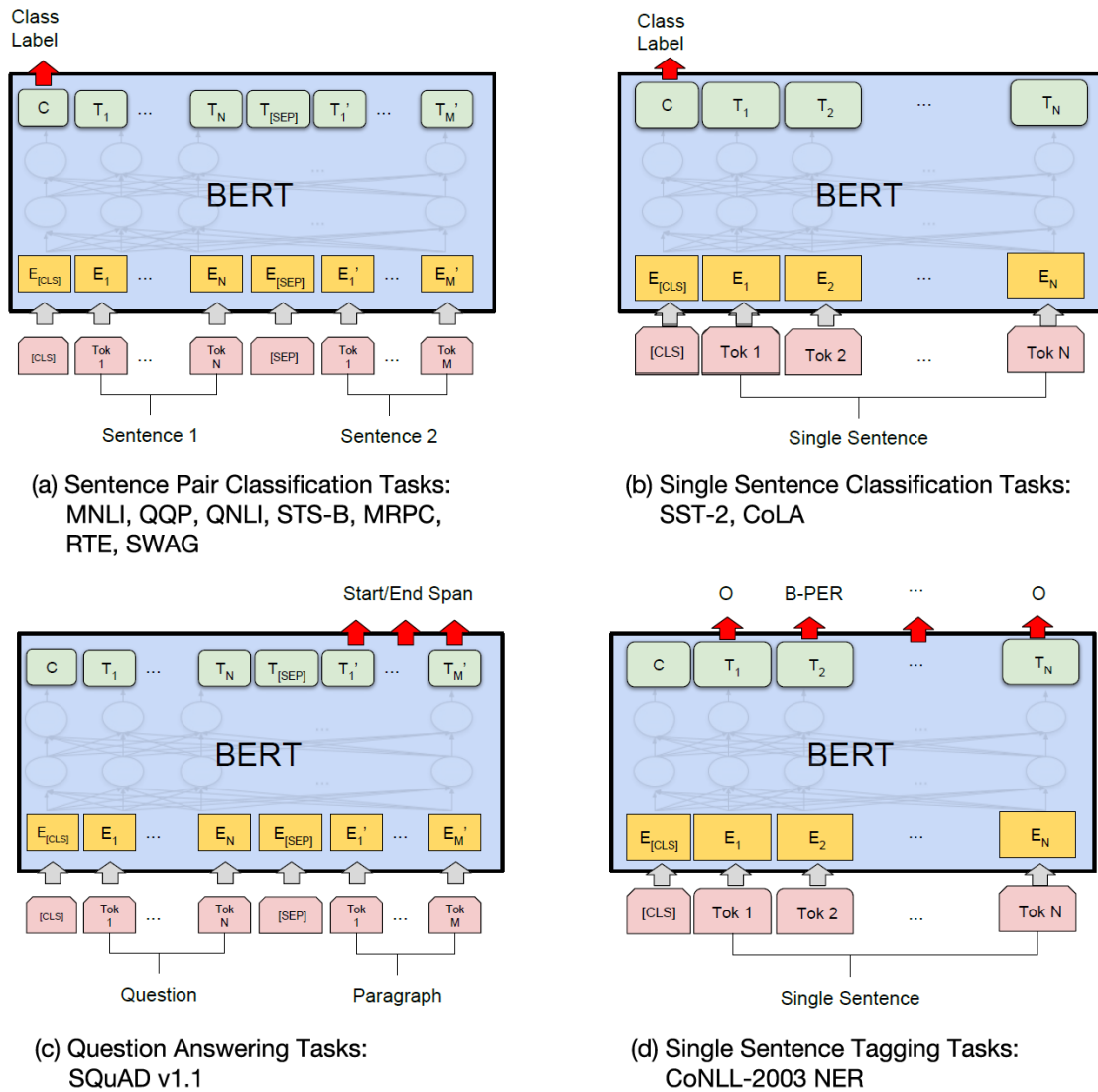


Figure 2.16: Training parameters added to the base model for fine-tuning BERT for different downstream task. Source (Devlin et al., 2019)

2.5 Works related to Relationship Extraction

RE was approached using a variety of machine learning paradigms mentioned above. In a supervised setup, the RE task was treated as a multiclass classification problem. Early research focused on extracting rich linguistic features to be used in feature-based approaches (Kambhatla (2004); Suchanek, Ifrim, and Weikum (2006); Rink and Harabagiu (2010)) such as POS tags, lexical gazetteers, WordNet representations, while some focused on kernel-based approaches using SVM (Chowdhury and Lavelli (2013); Thomas et al. (2013)), convolution tree kernel (Qian, Zhou, Kong, Zhu, & Qian, 2008) and dependency tree kernel (Bunescu & Mooney, 2005) for the classification task. However, the recent spike in deep neural networks has changed the focus to using deep networks to learn features rather than hand-crafting the features. Socher et al. (2012) worked on an RNN network to learn vectors in the constituent parse tree between target entities to determine their semantic relationships. Similarly, Ebrahimi and Dou (2015) built an RNN to obtain a classification representation based on the dependency path between the two target entities. Using a deep CNN, Zeng et al. (2014) extracted features at the lexical and sentence level and concatenated them to construct a feature vector for classification. Wang et al. (2016) used a CNN architecture with a multilevel attention mechanism to capture entity-specific and relation-specific attention with respect to target entities and relationship, respectively. Similarly, Shen and Huang (2016) used a CNN in the sentence to obtain a sentence convolution vector, which is then concatenated with an attention-based context vector to perform the RE task. Although the BERT model is not designed for RE tasks, Baldini Soares, FitzGerald, Ling, and Kwiatkowski (2019) and S. Wu and He (2019) attempted to perform the tasks by reformulating the input in BERT with special markers to highlight the target entities.

Among the many approaches attempted, considering the nature of the RE task, that is, to classify semantic relations between two entities, some researches focused on making use of dependency parse trees to extract features. Using dependency parse trees has shown to be effective as they are capable to capture long-range syntactic relationships which are unclear to observe from the surface form of the sentence. Classical feature-based models (Kambhatla (2004); Bobić, Fluck, and Hofmann-Apitius (2013); Thomas et al. (2013)) extracted lexical and syntactical features of tokens that lie in the dependency path between the two target entities and encoded along with other linguistic features of tokens in the sentence. Bunescu and Mooney (2005) showed the effectiveness of reducing the dependency parse tree to *shortest dependency path* between entities using a tree kernel approach. Miwa

and Bansal (2016) attempted to capture both the substructure information of the dependency parse tree and the word sequence by using a bidirectional sequential LSTM stacked with a bidirectional tree-structured LSTM.

In contrast to the techniques mentioned above, Y. Zhang, Qi, and Manning (2018) and D. Li and Ji (2019) attempted to model the RE task by applying a Graph Convolution Network (GCN) (see Section 3.1) in the dependency parse tree along with contextualized word vectors. Y. Zhang et al. (2018) used the GCN on a pruned dependency parse tree based on the target entities rather than restricting to the shortest dependency path. Contextualized word representation obtained from a bi-LSTM was used as input to the GCN architecture. This bi-LSTM is trained jointly with the rest of the network. Similarly, D. Li and Ji (2019) modeled a 2-layer GCN in a similar approach however obtaining contextualized word embedding from BioBERT (as they performed their task on DDI) embedded with positional encoding and used dependency parse tree of the entire sentence as input rather than pruning it.

Considering the nature of the RE task, and the capability of dependency parse tree in capturing semantic relationship between entities, this thesis focuses on a similar approach to a GCN architecture. Rather restricting the GCN architecture only to contextualized word embeddings, this thesis attempts to encode additional linguistic features of tokens in the dependency parse tree along with a representation of dependency path between entities using Node2Vec embeddings. More detailed description of the specific GCN architecture used along with the input representation is given in the following chapter (Chapter 3).

Chapter 3

Dependency parse trees and Graph Neural Networks

Human languages are organized in a way that captures the basic view of how humans observe the world (de Marneffe, Manning, Nivre, & Zeman, 2021). We see entities (or objects) participating in events (actions or states), and these entities and events have attributes that describe their nature. The dependency graphs try to capture this using three fundamental linguistic units, namely **nominal**, **clause**, and **modifier**. These three units can be phrases (consisting of multiple words) or minimally be just a single word.

Nominals resemble entities, and typically a nominal phrase will contain a noun as the head word of the phrase. **Clause** resembles events, and usually have a head word also known as **predicate**, mostly a verb, but possibly an adjective or adverb. **Modifiers** are used to describe the attributes of entities or events and have mainly an adjective or adverb as the head word. In a phrasal construct, a clause has a main predicate that expresses a state or action, and since most of the time these actions involve participants, they also contain nominals. These types of composition lead to a hierarchical structure where clauses can contain nominals, modifiers, and other clauses; nominals can also contain all three units; and modifiers can contain other modifiers. To express the hierarchical structure, the dependency grammar uses a construct **head - dependent**, where a phrase has a head, and the rest it contains are considered dependents of the head. This relationship is symbolized with a directed arrow pointing towards the dependents (more precisely, to the head of the dependents) starting from the head. This head-dependent binary asymmetrical relation results in a tree structure with the main predicate as the root.

As we have seen in earlier chapters, the relationship extraction task requires a triple annotation in the form of (relation, X, Y), where X and Y are two target entities, and the

relation depicts the relation between the entities. Since dependency parse trees are capable of providing approximations of semantic relationships between predicates and their arguments, this thesis focuses on generalizing over dependency parse trees to obtain latent path representations to distinguish different semantic connections between the target entities.

Link Prediction (LP) versus Relation Extraction (RE) Link prediction (LP) is a task that focuses on predicting whether two nodes in a graph network are likely to have a connection or not (M. Zhang & Chen, 2018). LP is applied to graph structured data, which are widely used in many domains such as social networks, recommender systems, knowledge graphs, proteins or gene interactions, etc. A practical example of an LP task on a social network graph would try to predict whether two people who are not currently connected will connect in the future or not. Graph Neural Networks (GNNs) (Kipf & Welling, 2017) are a family of neural approaches that are applied to graphs to address certain tasks, such as link prediction. Therefore, for the above social network example, a GNN approach uses the information about the network structure, i.e. connections that are common for given two nodes (humans), along with features of the nodes, such as hobbies, interests, age, etc. to predict whether or not the two nodes will have a connection in the future.

Relation extraction (RE) tasks can be perceived in the same way as LP. Similarly to the two target nodes in the LP task, in RE there are two target entities (E_1 and E_2). In LP, the graph structure encapsulating the target nodes along with the features associated with it is used to predict the existence of a link, whereas in RE one can leverage upon the dependency parse tree (which is a type of graph) and use token features (such as token embedding, POS tag, etc.) to predict the relationship between the two entities. However, in contrast to LP, where the prediction is the existence of a (physical) link, in RE it would be a semantic link. In a multiclass setting, this link will also have a type rather than just a prediction of whether there exists a link or not. Since the LP and RE task share similarities, this thesis attempts to model the RE task using the GNN approach, which is discussed in the following section.

3.1 Graph Neural networks (GNN)

GNN operates on graph data. A graph is a structured way to represent data. Many natural phenomena or systems and their interactions, such as social networks, molecules, organizations, citations, transactions, etc., can be naturally represented as graphs. Therefore, formally, a graph G can be defined as a set of vertices/nodes V and a set of edges E that connect them. Each vertex $v \in V$, can have one or many properties, such as a

person vertex on the social network, has features such as age, sex, name, etc. The edges represent the relationship between these vertices. They can be directed (similar to dependency parse trees) or undirected (similar to social network connections) and may or may not contain properties similar to vertices. Unlike other data types, such as image data or text data, graph data lack a fixed structure and ordering. Sentences are composed of a specific ordering of words, and image data is composed of a grid of pixels. Ordinary neural network-based approaches struggle in processing graph structured data, leading to a separate family of neural approaches called graph neural networks (GNN) (Bruna, Zaremba, Szlam, and Lecun (2014); Defferrard, Bresson, and Vandergheynst (2016); Duvenaud et al. (2015); Atwood and Towsley (2016); Kipf and Welling (2017)).

The basics of GNN (Kipf & Welling, 2017) are very similar to the basics of Convolutional Neural Networks (CNN) (LeCun et al., 1999). CNN on image data takes each pixel and extracts information about its region by aggregating the pixel within the context of its neighborhood. The next layer then extracts information about that region in the context of its neighboring regions. Doing this over enough number of layers enables the network to be able to reason over different parts of the whole image, and passing this over some linear computations will identify the relevant object.

Similarly, relaxing the properties of fixed structure and ordering as in an image data, GNN performs convolution by passing messages from nodes to its neighbors. This is achieved by aggregating the messages received from all neighboring nodes. Performing this over multiple rounds enables more and more information to be passed around the graph. After a sufficient round of message passing, the GNN obtains a final representation of each node in the graph that describes it given the larger context. Each round of message passing can be viewed as a series of layers, just as in CNN. The final representation of each node can be considered as a node embedding, which can be used as input to any ordinary neural network-based architecture to perform the classification task.

In the family of GNN techniques, many different approaches are currently being explored. Since motivation to investigate the link prediction-based approach was obtained from M. Zhang and Chen (2018), this thesis experiments with their approach using a Deep Graph Convolution Neural Network (DGCNN).

3.1.1 Deep Graph Convolution Neural Network (DGCNN)

Among the different implementations of GNN, this thesis focuses on a specific architecture, DGCNN (M. Zhang et al., 2018). Apart from the reason it was used in the LP

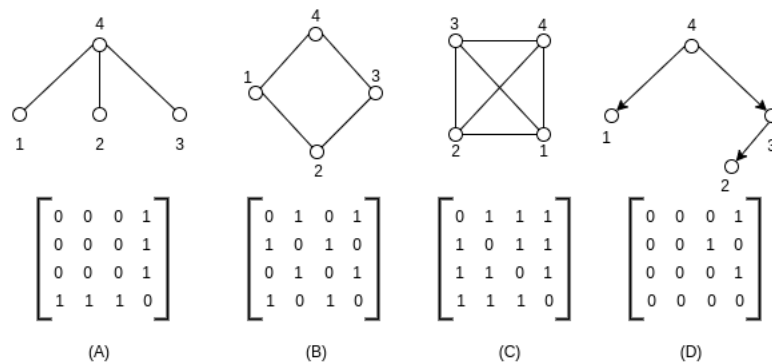


Figure 3.1: Graphs and adjacency matrices

article (M. Zhang & Chen, 2018) as mentioned above, DGCNN obtained SOTA (state-of-the-art) performance in two of the benchmark bio-informatics datasets, such as PROTEINS (Borgwardt et al., 2005) and D&D (Dobson & Doig, 2003) and two social network datasets COLLAB (Yanardag & Vishwanathan, 2015) and IMDB-M (Yanardag & Vishwanathan, 2015). Most importantly, the architecture was designed to accept graph input of arbitrary size, which was relevant to this thesis as the size of the input vary sample to sample.

Similarly to the basic GNN architecture (Kipf & Welling, 2017), the input to DGCNN is represented using two matrices. One is the Adjacency Matrix \mathbf{A} and the other is the Information Matrix (IM); for simplicity of representation in the formula, the notation \mathbf{X} will be used instead of IM in this chapter.

Adjacency matrix (\mathbf{A}): Also known as the connection matrix is a square matrix $A \in \mathbb{R}^{n \times n}$. Its dimension corresponds to the number of vertices (n) on the graph, with 1 or 0 in position of (v_i, v_j) depending on whether v_i is adjacent to v_j or not. For a graph without self-loops, the matrix will have a diagonal with 0 s. For an undirected graph, the A matrix will be symmetric. Figure 3.1 shows a few simple examples of graphs and how they are interpreted in an A matrix.

Since dependency parse trees are directed trees, the resulting A matrix is not symmetrical, as shown in Figure 3.1.D. Furthermore, modifications to the values within the A matrix can be made by providing weight values instead of binary values to show the strength of the connection each node can have with its neighboring nodes. Although this is a possibility of a modification that could be tested, this thesis will not take this discussion further, as it is beyond the scope of this thesis.

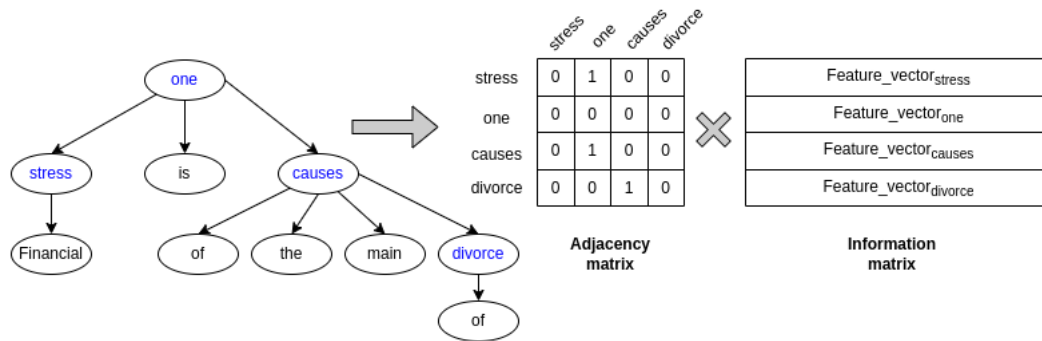


Figure 3.2: Representation of an Adjacency matrix and corresponding Information matrix for entity pair (*stress*, *divorce*)

Information matrix (IM/X): Unlike vertices in a typical graph structure, vertices in the dependency parse tree contain rich linguistic features such as token embedding, POS tags, gazetteer encoding, etc. Therefore, to capitalize on these features and their connections to other nodes, we create an Information Matrix $X \in \mathbb{R}^{n \times c}$, where n refers to the number of nodes within a graph, and c refers to the size of the feature vector of each node in the graph. Therefore, for each row in the A matrix, a corresponding row in the X matrix (see Figure 3.2) will be created that contains the feature vector of each node in the graph.

The specifics of the matrix IM (or X), such as how it is created and what features are encoded there, will be discussed later in this chapter in Section 3.3. The following sections will focus on how these matrices (A , X) are used to create a graph representation using the DGCNN architecture. As shown in Figure 3.3 DGCNN consists of four sequential stages:

- (1) **Graph Convolution layers:** Extract local substructure features of vertices and define a consistent vertex ordering
- (2) **Sort Pooling layers:** sort vertex features and normalize input sizes
- (3) **1D Convolution network:** extract abstract features and create graph representation
- (4) **Dense layer:** make predictions based on graph representations.

3.1.1.1 Graph Convolution layers

In a nutshell, the graph convolution layers perform message-passing from all nodes to their neighbors, so that at the end of multiple rounds of message passing, each node will

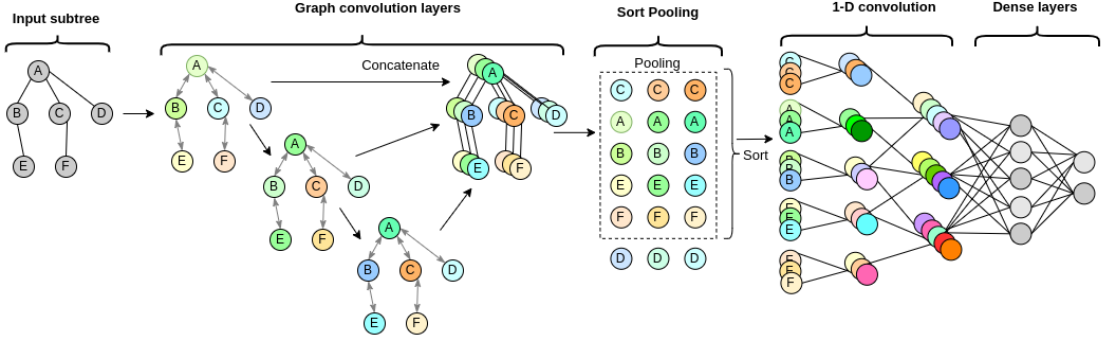


Figure 3.3: Illustration of DGCNN architecture; Source: [M. Zhang et al. \(2018\)](#)

have a representation given the larger context (the graph). Given a graph G , represented by the matrices \mathbf{A} and \mathbf{X} , the graph convolution is performed as follows:

$$Z = f(\tilde{D}^{-1}\tilde{A}XW) \quad (3.1)$$

where $\tilde{A} = A + I$ with I as the identity matrix to add self-loops to the A matrix. \tilde{D} is a diagonal degree matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, $W \in \mathbb{R}^{c \times c'}$ is matrix of trainable graph convolution parameters, f is a nonlinear activation function and $Z \in \mathbb{R}^{n \times c'}$ is the output of the activation function.

For a deeper understanding of the graph convolution, it can be split into 4 steps. The first step is the linear transformation of the node information matrix by \mathbf{XW} , mapping c feature vectors to c' vector; here the filter weight \mathbf{W} is shared among all vertices. The second step is the propagation of the node information to the neighboring vertices, as well as the node itself, by $\tilde{A}Y$, where $Y = XW$. For further understanding, this can be broken down as $(\tilde{A}Y)_i = \sum_j \tilde{A}_{ij}Y_j = Y_i + \sum_{j \in \Gamma(i)} Y_j$ or, that is, the row i of the resulting matrix is the sum of Y_i itself and Y_j from neighboring nodes of i ($\Gamma(i)$). As the third step, to maintain a fixed feature scale after graph convolution, each row i is normalized by multiplying by \tilde{D}_{ii}^{-1} . The final step is to apply a point-wise non-linear activation function f and output the results of the graph convolution.

Similarly to a traditional convolution kernel (see Section 2.2), a graph convolution extracts information about the local substructure by aggregating information about the vertex in local neighborhoods. Therefore, to extract features of multiscaled substructures, multiple layers of graph convolution (such as Equation 3.1) can be stacked as follows:

$$Z^{t+1} = f(\tilde{D}^{-1}\tilde{A}Z^tW^t) \quad (3.2)$$

where $Z^0 = X$, $Z^t \in \mathbb{R}^{n \times c_t}$ is the output of the graph convolution layer t and $W^t \in \mathbb{R}^{c_t \times c_{t+1}}$ maps the c_t feature vectors to the c_{t+1} feature vector. After multiple layers of graph convolution, an additional layer is added to horizontally concatenate the generated output Z^t , where $t = 1, \dots, h$ resulting in an output $Z^{1:h} := [Z^1, \dots, Z^h]$, where h is the number of graph convolutions applied and $Z^{1:h} \in \mathbb{R}^{n \times \sum_1^h c_t}$. Therefore, in the concatenated output $Z^{1:h}$, each row i contains the encoding of multiscale local substructure information of vertex v_i , which will be addressed as the “**feature descriptor**” of each vertex.

3.1.1.2 Sort Pooling layers

The main objective of the SortPooling layer is to sort the feature descriptors created in the previous step in a consistent order before using them as input into a traditional 1D convolutional and dense neural network. [M. Zhang et al. \(2018\)](#) consider this step to be one of their novel approaches in their proposed DGCNN architecture.

In this layer, given an input $Z^{1:h}$, which is a tensor of dimension $n \times \sum_1^h c_t$, the output is produced as a tensor (Z^{sp}) of dimension $k \times \sum_1^h c_t$, where k is an integer defined by the user. This is accomplished in two steps. The first step is to perform a row-wise sorting of the input $Z^{1:h}$ based on Z^h , since Z^h is the last layer of convolution, considered the output of the vertices with the most refined substructure information. The vertex order sorting based on Z^h is performed by first sorting the last channel of Z^h in descending order. If two channels of two vertices have the same value, then the tie is broken by comparing the second to the last channel, and so on. If ties still exist, then the comparison will continue with the values produced by earlier layers, such as Z_i^{h-1}, Z_i^{h-2} , and so on until the ties are broken. [M. Zhang et al. \(2018\)](#) compares this ordering similar to lexicographical ordering but in reverse order (from right to left), which maintains a consistent ordering throughout all graphs, enabling to feed into a traditional neural network to perform the classification task.

Once the row-wise sorting is performed, the second step in the SortPooling layer is to normalize the size of the output tensor. This step is crucial, since the traditional neural network works with a fixed input size, and the input graph to the DGCNN system is not restricted to any fixed size (the number of vertices in the graph can vary). Without normalization, the output generated from the SortPooling layer will be a tensor of varying size, which the next layer cannot handle. Therefore, the first dimension of the output tensor is normalized to size k by truncating or extending the size from n to k . This is performed by deleting the last $n - k$ rows if $n > k$, or adding $k - n$ rows with zeros if $n < k$.

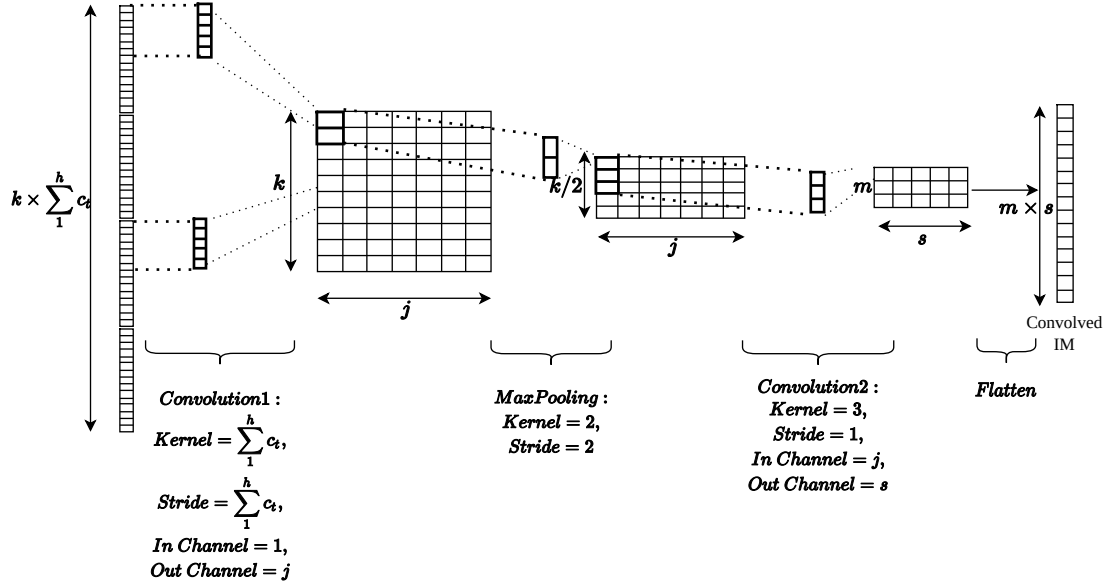


Figure 3.4: Architecture of 1D convolutional neural network in DGCNN

3.1.1.3 1D Convolution Neural Network (1D-CNN)

After the SortPooling layer, the output Z^{sp} generated is of size $k \times \sum_1^h c_t$. To train a 1D CNN on them, Z^{sp} is first reshaped row-wise (by concatenating the rows one after the other) into a vector $k(\sum_1^h c_t) \times 1$. This step is then followed by a 1D convolution layer (see Figure 3.4) with a filter size and a step size equal to $\sum_1^h c_t$, to sequentially apply filters on the feature descriptors of the vertices. This is then followed by a Max-pooling layer with a 2 by 2 window size and stride size 1, to extract the prominent features. This is then followed by another 1D convolution layer to learn local patterns in the node sequence. The final representation is then flattened to produce a single-dimensional vector, which will then be used as a graph representation for classification.

3.1.1.4 Dense Layer

The graph representation obtained from the 1D convolution network is used as input to a fully connected neural network with a hidden layer and a softmax layer to perform the classification task. The cross entropy loss function¹ with class weights passed as a parameter was used to calculate the loss and backpropagate to fine-tune the weight parameters of all layers in DGCNN. Here, the class weights were calculated by obtaining the inverse of the total number of data samples observed in the training data ($1/\text{total_num_datapoints_within_class}$)

¹<https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>

to address the imbalanced data distribution in the corpus.

3.2 Preprocessing text

As with any language-related task, a text document goes through a series of preprocessing steps to clean and extract linguistic features to perform the task. The process of cleaning and extracting has a considerable amount of influence on the overall performance of the system. A typical NLP preprocessing steps include sentence splitting, tokenization, sometimes followed by task-relevant Gazetteer annotation, possibly Named-Entity-Recognition(NER), part-of-speech (POS) tagging, and dependency parsing. These pre-processing steps are so common that there exist many different packages to facilitate this task using different approaches (see Section 2.3). However, for this thesis, all pre-processing steps were performed using the GATE(Cunningham et al., 2013) framework.

3.2.1 Pre processing pipeline

The three data sets (DDI, Sem10, and MeasEval) were processed with a series of GATE (Cunningham et al., 2013) modules: ANNIE Tokenizer, ANNIE Sentence Splitter, Stanford Parser for POS tagging (Klein & Manning, 2003) and Stanford Dependency Parser (de Marneffe et al., 2006). Errors in the earlier modules have a snowball effect on the following modules by propagating errors, resulting in incorrect annotations. For example, a wrong tokenization will affect the POS assigned to it, possibly leading to a wrong dependency parse tree. To prevent the snowball effect of errors throughout the pipeline and to obtain an optimal dependency parse tree, a sequence of additional JAPE (GATE’s rule description language) rules were added to post-process certain outcomes of the modules. The following is a list of JAPE rules added to fix certain unfavorable outcomes observed by the pipeline modules.

- **Mixed character protection:** This rule prevents splitting tokens composed of different character types into different tokens (e.g. δ 13CTOC \nrightarrow δ , 13, CTOC) while preserving the usual behavior of ANNIE tokenization when dealing with mathematical symbols. (for example “ $5 \leq 2\theta / \circ \leq 80$ ” \rightarrow “5”, “ \leq ”, “ 2θ ”, “/”, “ \circ ”, “ \leq ”, “80”).
- **Number normalization:** This rule prevents the default behaviour of ANNIE tokenizer splitting decimal numbers into three different tokens (256.89 \nrightarrow “256”, “.”, “89”). Also, it identified words that resemble numbers as numbers. This was accomplished using the Number Normalizer GATE plugin, developed by a previous student from the CLaC lab to modify the behavior of the ANNIE tokenizer. However, an

additional rule was added for an unhandled case where decimal numbers such as *.005* without a preceding integer before full-stop being split into two (*.005* ↦ “.”, “005”).

- **Abbreviation period protection:** Although the ANNIE sentence splitter preserves abbreviation periods without splitting them into sentences, scientific documents contain some additional abbreviations which are not commonly found in English text, such as *Fig.*, *sp.*, *spp.*, *Tab.* etc. To fix this issue, instead of explicitly specifying what those abbreviations are, a simple pattern-based JAPE rule was added to fix this issue. The rule utilized (in regular expression) is as follows:

$$\backslash.(\backslash) | \backslash S+ [a-z])$$

In simple words, if there exists a full-stop punctuation mark followed by ending parentheses or followed by one or more spaces, with a preceding word starting with a lowercase character, then the full-stop punctuation mark is considered as a punctuation mark to abbreviate a word rather than the ending of a sentence.

- **List and interval protection:** Scientific articles frequently report on intervals expressed in different ways and with varying lengths (e.g., “34-40” or “265.32, 272.30, and 340.21”). These lists, as a whole, play a role in the text. Unfortunately, they do not receive proper dependency-parse assignments. To improve dependency relationship assignments, a JAPE rule was introduced to group cardinal number tokens (“CD”) belonging to a list into one token and manually assign the POS tag “CD” to the grouping. The following is a summary of the grouping patterns used for this purpose.

$$\begin{aligned} & \text{CD (: | - | to) CD} \\ & \text{CD (, CD)* and CD} \end{aligned}$$

Once the data sets are preprocessed through GATE, the linguistic features such as token offsets, sentence offset, POS tags of each token in a sentence, and dependency parse trees are extracted in a CSV (comma separated value) file by utilizing GATE’s *Configurable Exporter* module for further processing to create the input representation for my experiment models.

3.3 Constructing Information Matrix (IM)

As explained in Section 3.1.1 the input to the DGCNN is made up of two matrices: Adjacency matrix(*A*) and Information matrix(*IM*). The *A* matrix represents the connectivity of entities based on the dependency graph, while the *IM* matrix contains the feature vectors

of each token/node within the graph. **Feature vector** of each token/node is constructed by concatenating multiple linguistic features. Since the thesis addresses three different data sets to evaluate the impact of linguistic features on the RE task and the achievement of SOTA performance was not part of the objective, the features extracted to represent in *IM* are basic and can be produced by any text processing library (e.g. SpaCy or NLTK) or frameworks (e.g. GATE) than task-specific.

According to the paper on Link Prediction (M. Zhang & Chen, 2018), the authors extract a subgraph surrounding the target nodes and construct the *A* and *IM* to represent them. This subgraph serves as a context to determine whether there exists a link between the two target nodes or not. Similarly, I constructed the context by extracting all nodes on the shortest path between the two target entities, as they have been proven to be effective (Bunescu and Mooney (2005)). Therefore, as a first step, before constructing feature vector representations for tokens/nodes, the target entities (E_1, E_2) were first identified, and then the subtree² encompassed by the target entities were extracted from the dependency parse tree. Since there are 3 data sets, which are different from each other, the following sections will discuss the steps to identify the entities (E_1, E_2) individually for each data set.

3.3.1 Target entity pair extraction and class labeling for DDI

Summary of the task: The DDI (Segura-Bedmar et al., 2013) corpus consists of documents describing drug-drug interactions extracted from the DrugBank (Wishart et al., 2006) database and MEDLINE abstracts. Each document consists of multiple sentences, where each sentence is provided with a unique sentence ID along with all drugs within the sentence identified as **Entities** and explicitly annotated by the task organizers. For sentences containing more than one entity, they will contain **Pair** annotations, which denote the pair of entities between which a relationship should be predicted. The pair annotation for each sentence is created by taking all the permutations of the identified entities in a sentence (that is, a sentence containing 4 entities will have 6 pair annotations, as shown in Ex 7). However, the order of the entities within the pair annotation is based on sentence ordering. (that is, E_1 should appear before E_2 in the sentence).

Ex 7: “Anakinra₁: Concurrent administration of anakinra₂ (an interleukin-1 antagonist) and another TNF-blocking agent has been associated with an increased risk of serious infections, an increased risk of neutropenia and no additional benefit compared to these medicinal products alone.”

²the term *subtree* is used instead of *subgraph*, since a tree is a kind of graph and the dependency parse tree has a tree-like structure

DDI (Anakinra₁, anakinra₂): False
DDI (Anakinra₁, interleukin-1 antagonist): False
DDI (Anakinra₁, TNF-blocking): False
DDI (anakinra₂, interleukin-1 antagonist): False
DDI (anakinra₂, TNF-blocking): True (type: Effect)
DDI (interleukin-1 antagonist, TNF-blocking): True (type: Effect)

The drug that has a positive interaction with another drug will have a "type" of interaction as additional information; the type can be one of the four types of interactions the drugs could have between them: Advise, Effect, Mechanism, and Interaction. However, to keep the task simple, only the interaction (True or False) was predicted instead of trying to predict the type of interaction.

As a preliminary step to construct the representation IM , the target entities between which a relationship should be predicted (E_1, E_2) are extracted from the corpus through the following steps:

Selection of sentence: Given the DDI corpus, sentences containing the annotation **Pair** were only considered and pre-processed, to be used to extract input data to train the models, the rest of the sentences in the corpus were discarded.

Selection of token pairs (E_1, E_2): Since DDI explicitly provides the target entities (E_1, E_2) in the form of annotation *Pair*, this information is used directly to create the token pairs necessary to construct the IM . However, the entities annotated in the corpus may sometimes be a composition of multiple tokens (e.g., *HIV Protease Inhibitors, interleukin-1 antagonist* (as shown in Ex 7)), in this case only one token is chosen given the span to represent the pair of tokens (E_1, E_2). This is due to the fact that the dependency-parse tree is made of tokens rather than spans. To choose a single token given the span of the entity, I attempted to find the head noun following the hypothesis, as the token should be a noun (with a POS tag *NN/ NNS/ NNP/ NNPS*) and if there exist many nouns within the span, the last token of nouns will be chosen as the token to represent the entity.

Labeling token pairs: Each token pair is assigned the label provided by the organizers. False classes received a label 0 and true classes received a label 1.

3.3.2 Target entity pair extraction and class labeling for Sem10

Summary of the task: SemEval-2018 task 8 (Hendrickx et al., 2010) is focused on the extraction of the semantic relationship between pairs of nominal given a sentence. Rather

than restricting the sentences to a specific domain, this task focuses on extracting generic relationships based on the practical interest that can be observed in the English text. This task was organized as a multiway classification task, where a relationship should be predicted for a pair of entities out of 10 relationship defined, including “Other” (no-relationship class).

Selection of sentence: In contrast to the other two data sets, the Sem10 is a more engineered dataset where the corpus consists of sentences with a pair of nominals explicitly annotated for each sentence. Therefore, all sentences in the corpus were considered training data.

Selection of token pairs (E_1, E_2): Similar to the DDI task, the pairs of target entities were explicitly annotated for each sentence in the corpus. However, the entities annotated in the corpus might sometimes be a composition of multiple tokens (e.g., *science fiction*, *health care provider*). Therefore, to extract a single token as Entity, the head noun is chosen following a similar approach as DDI mentioned above.

Labeling token pairs: Each pair of token annotated in the corpus is given a single class out of the 10 classes defined by the organizers. The 10 classes are *Cause_Effect*, *Instrument_Agency*, *Product_Producer*, *Content_Container*, *Entity-Origin*, *Entity_Destination*, *Component_Whole*, *Member_Collection*, *Message_Topic*, and *Other*. The definitions of the classes are provided in Section 1.2.

3.3.3 Target entity pair extraction and class labeling for MeasEval

Summary of the task: The MeasEval(Harper et al., 2021) task consists of 5 subtasks (non-independent) that cover entity detection, span detection, and relation extraction across multiple sentences. Since the main objective of the task is to identify *Quantities (Q)* and its related entities, the term “**Annotation set**” (**AnnSet**) is used to define all related entities and relations associated with a particular Q. The individual entities within an AnnSet will be identified as “**Annotation type**” (**AnnType**). A basic annotation set can consist of 4 annotation types (Quantity (Q), MeasuredEntity (ME), MeasuredProperty (MP) and Qualifier (QL)) and 3 relation types (HasQuantity (HQ), HasProperty (HP), and HasQualifier(HQ)).

Unlike the other data sets, the MeasEval dataset comes with its own set of challenges. With limited data (show in Table 3.1 and 3.2), wide range of topics involved in the dataset, and the idiosyncrasies of vocabulary used in each scientific field makes the task difficult.

Table 3.1: Training data: Frequencies of Annotation types

Annotation	Frequency
Quantity (QA)	866
MeasuredEntity (ME)	855
MeasuredProperty (MP)	544
Qualifier(QL)	238

Table 3.2: Frequencies of different composition of Annotation Sets

Composition	Training	Testing
Q,MP,ME	365	152
Q,ME	281	99
Q,MP,ME,QL	148	86
Q,ME,QL	34	23
Q,MP,ME,QL,QL	25	9
Q	6	
Q,MP	4	
Q,MP,QL	1	
Q,ME,QL,QL	1	
Q,MP,ME,QL,QL,QL	1	

Also the complexity of the task increases with the multiple phases having to be achieved to complete the task, such as:

- Identification of AnnTypes (Q, ME, MP and QL); There are a few cases where a particular span of text will have different AnnType for different AnnSet, as shown in Ex 8.

Ex 8: “The SCGs show that from the initial version barcode-0.90 there were two(Q₁) coherent clusters (X) in the system. The smaller one is around 10%(Q₂) of the code(ME₂) while the larger is around 40%(Q₃) of the code(ME₃).”

Annotation Set-1: (Q₁, ME(X))

Annotation Set-2: (Q₂, ME₂, MP(X))

Annotation Set-3: (Q₃, ME₃, MP(X))

And more cases where a particular AnnType belonging to more than one AnnSets (i.e. a span identified as MP can belong to both annotation set 1 and 2) as shown in Ex 9.

Ex 9: “We recruited 147 patients from consanguineous pedigrees(ME) (parents of the proband were second cousins or more closely related) who were diagnosed with permanent diabetes (MP) before 6 months(Q₁) of age or were diagnosed with permanent diabetes before 9 months”

RB , DT NNS VBD IN NN CD MD VB VBN TO VB IN RB JJ NNS IN JJR CD NN
 However , all elements included within Fig. 5 can be considered to be at very low concentrations of < 2 ppm

Figure 3.5: Tokenized and POS tagged sample sentence from the MeasEval data set

(Q₂) and had any additional clinical features that made a diagnosis of type 1 diabetes less likely.”

Annotation Set-1: (Q₁, ME, MP)

Annotation Set-2: (Q₂, ME, MP)

- Determining the span of the entity (see Table 3.3 for a comprehensive list of the range of span sizes with examples with respect to each annotation type)
- Classification of Q to detect modifiers and detection of units if exists any.
- Grouping annotation types into annotation sets; there are rare cases where the annotation set can span multiple sentences (as shown in Ex 8 above).

Since this thesis only focuses on relation extraction, additional tasks such as detecting modifiers and units were excluded from experimentation. Also the detection of QL AnnType was excluded as it lacked the definition as to what it corresponds to.

Unlike the other two data sets, in MeasEval the entities were not explicitly annotated. Therefore a system has to perform Entity detection prior to performing a relation extraction. Considering all these factors, I made certain assumptions to determine what the target entities could be, prior to extracting the subtrees. Based on the sentence in Figure 3.5, the assumptions and the intuition behind them are as follows:

- (1) Entity spans are unknown. In other words, given a sentence one do not know whether there exist any token that falls into any category (Q, ME or MP). Hence entities should to detected first.
- (2) However, based on the guideline provided, Q always serves as the base for a given AnnSet.
- (3) Even though a Q span is not explicitly given, based on the definition, a Q span should contain a numerical value. This numerical value will have a POS tag *CD*. For the ease of readability the term *base_CD* will be used to address this token.

Assumption 1: A Quantity should have a POS tag CD

Based on the sentence in Figure 3.5, “5” and “2” are possible candidates for base_CD.

Table 3.3: Word count within spans for each annotation type. Spans are split by space to obtain an approximate word count in given gold data.

Annotation type	word count	occurrence count	Example
Quantity	1	427	“Five”
	2	440	“4 days”
	3	144	“After 8 days”
	4	75	“less than one day”
	5	43	“up to at least 3Rp”
	6	20	“1.5 mm and the 8.5 mm”
	7	10	“a few percent to around thirty percent”
	9	4	“increases from 2.6 km s ⁻¹ to 25 km s ⁻¹ ”
	10	1	“(a) 1.3 (b) 2.4, and (c) 2.2 mg cm ⁻² ”
	Measured Entity	1	430
2		397	“size fractions”
3		133	“sets of cases”
4		78	“input of odd nitrogen”
5		55	“diatom silicon isotope fractionation factor”
6		18	“cells of all three germ layers”
7		12	“impactors have churned the soil on Mars”
8		7	“free-air enrichment by supplementary CO ₂ in field plots”
9		6	“Refined crystallographic parameters for (1) from PXD and PND”
10		4	“ $\delta^{30}Si$ values of size fractions between 2 and 20 μm ”
11		4	“deficient HA start their decomposition at temperatures lower than pure HA”
12		1	“pure HA (ratio CaP=1.67) is stable in an air and argon atmosphere”
13		2	“North Atlantic igneous province (NAIP), the Kilda Basin, and the northern rain belt”
14		1	“Tropical <i>E. grandis</i> × <i>E. urophylla</i> expressing a stress-inducible rd29a promoter-CBF2 transcription factor cassette”
Measured Property	1	400	“thick”
	2	167	“spinner flask”
	3	88	“taken any medication”
	4	37	“storage in the dark”
	5	23	“maximum yield for the year”
	6	10	“fractional integrated differential brightness of BS1”
	7	3	“gelatin-coated tissue culture plates in differentiation medium”
	8	3	“rapid cooling to ambient temperature at a rate”
	9	5	“self-pick up process has been shown to lead to”
	10	1	“found to carry the GFP-Neo cassette in the SOX2 locus”
	11	2	“Seismic amplitude differences between the 38 °C and 34 °C scenarios”
	12	2	“produce the same backward slice as well as the same forward slice”
	14	1	“category-4 still make up the second largest group and when added together with category-2”

Table 3.4: Continuation of Table 3.3... Word count within spans for each annotation type. Spans are split by space to obtain an approximate word count in given gold data.

Annotation type	word count	occurrence count	Example
Qualifier	1	107	“mean”
	2	52	“cooler by”
	3	50	“high magnetic susceptibility”
	4	31	“random high-angle grain boundaries”
	5	17	“Compared to the C1 model”
	6	15	“better than in the rotating mode ”
	7	13	“average solar XUV flux at 0.047 AU”
	8	8	“at the location of the accelerated electron signature”
	9	8	“(at T = 1200 K) for $\Sigma 5$ grain boundaries”
	10	4	“post-inoculation of single suspension BC1 cells in ultra-low attachment plates”
	11	1	“accumulated over 3 byr and mixed into 1.5–2.6 m of soil”
	17	2	“inferred from elemental abundances measured at various locations on Mars by Spirit, Opportunity, Pathfinder, and Viking Landers”
	19	1	“little evidence to suggest that standard clinical information on preclinical vascular risk status would be helpful in the prediction”

- (4) But not all token with POS tag *CD* are quantities; scientific documents contain many mentions of numerical representations which are not exactly quantities depending on its context. Year of publication (*e.g. ... Peters et al., 2018...*) or section/table/figure numbers (*e.g. ... as shown in table 3.4...*) or scientific names with numerical suffixes which might not be explicitly attached with a symbol with the string prefixes (*e.g. ...across SD 1 and SD 2 are shown...*) are few examples for numerical values not being quantities in the given corpus.

Assumption 2: Not all “*CD*”s are Quantities

Based on Figure 3.5 “5” is a number of the figure and does not fall a Q span.

- (5) Figure 3.6 shows the gold annotation provided by the organizers for the sentence in Figure 3.5. As per the example, ME (“*elements*”) is approximately 16 tokens away (based on GATE tokenization) from Q (“*<2 ppm*”). However based on the dependency parse tree of the sentence as shown in Figure 3.6, ME is only 5 edge(hop) distance away from Q(“*2*”), when traversing along the path of the edges of the dependency graph. This shows that any AnnType within an AnnSet lies in close proximity (in terms of dependency parse tree) to Q, i.e. ME or MP lies semantically closer to Q.

Assumption 3: Different annotation types such as ME, MP are in close proximity to Q in terms of dependency parse trees (i.e. semantically closer)

Even though most cases of the AnnTypes of a given AnnSet are within a single sentence, there exist some cases in the corpus where the AnnSet spans multiple sentences. In other words, for a particular Q, its ME can be 2 sentences apart from Q. Since this thesis focuses on the dependency parse tree as a base to construct the input representation, AnnSets that lies within a sentence are only considered.

Given the above assumptions, the target entities (E_1, E_2) are extracted and assigned a label through the following steps:

Selection of sentence: From the resulting sentences after pre-processing the documents, sentences with token POS tagged as “*CD*”(base_CD) are processed to extract entities and the rest of the sentences are discarded.

Selection of token pairs (E_1, E_2): Based on *Assumption-1*, E_1 is always a token with the POS tag “*CD*” (base_CD), which pairs with other tokens in the dependency parse tree.

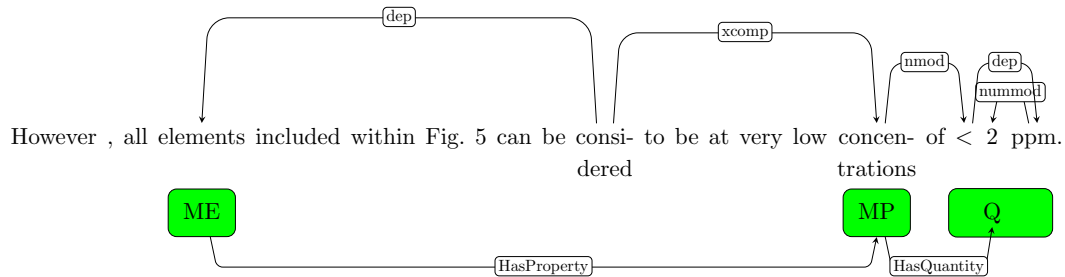


Figure 3.6: Gold annotation and dependency-parse tree representation of the sentence in Figure 3.5

In principle, all nodes in the dependency parse tree are candidates for E_2 . However, based on *Assumption-3*, I limit the number of nodes for E_2 based on the length of the connecting path. This results in multiple pairs for each base_CD.

For example, based on the dependency parse tree in Figure 3.6, E_1 will be the token/node with value “2”. If the maximum connection path length is set to 5, all token pairs that would be generated are as follows.

- Path length 1: (2, ppm)
- Path length 2: (2, <),
- Path length 3: (2, of), (2, concentrations)
- Path length 4: (2,considered), (2,to), (2,at), (2,be), (2,low)
- Path length 5: (2,elements), (2,5), (2,be), (2,can), (2,very)

Since the sentence in Figure 3.5 has another token “CD” “5”, it will also be considered as E_1 and will be paired with its neighboring nodes in the same way as shown above.

Labeling token pairs: Since the MeasEval task was approached in such a way that it would be possible to perform the detection of the entity and the extraction of the relation simultaneously, it was treated as a multiclass classification problem where the classes were constructed to reveal the type of entity and the relationship between entities. 1 out of 5 classes were assigned to each pair of tokens (E_1, E_2) based on the gold span into which each token falls in the given training gold annotation:

- Class 0 (Other): E_1 is not within a gold Quantity span
- Class 1 (Q-O): E_1 is within a gold quantity span, but E_2 is not within any gold spans
- Class 2 (Q-Q): E_1 and E_2 are within a gold Quantity span.

Table 3.5: Class labels assigned to token pairs generated for base_CD “2” on Figure 3.5

Token Pair	Class	Explanation
(2, ppm)	2 (Q-Q)	“2” and “ppm” fall in the gold Q span
(2, <)	2 (Q-Q)	“2” and “<” fall in the gold Q span
(2, of)	1 (Q-O)	“2” is in Q span but “of” is not within any span
(2, concentrations)	4 (Q-MP)	“2” is in Q span and “concentrations” is within MP span
(2, considered)	1 (Q-O)	“2” is in Q span but “considered” is not within any span
(2, to)	1 (Q-O)	“2” is in Q span but “to” is not within any span
(2, at)	1 (Q-O)	“2” is in Q span but “at” is not within any span
(2, be)	1 (Q-O)	“2” is in Q span but “be” is not within any span
(2, low)	1 (Q-O)	“2” is in Q span but “low” is not within any span
(2, elements)	3 (Q-ME)	“2” is in Q span and “elements” is within ME span
(2, 5)	1 (Q-O)	“2” is in Q span but “5” is not within any span
(2, be)	1 (Q-O)	“2” is in Q span but “be” is not within any span
(2, can)	1 (Q-O)	“2” is in Q span but “can” is not within any span
(2, very)	1 (Q-O)	“2” is in Q span but “very” is not within any span

- Class 3 (Q-ME): E_1 is within a gold quantity span and E_2 is within a gold Measure-Entity span
- Class 4 (Q-MP): E_1 is within a gold quantity span and E_2 is within a gold Measured-Property span

Using these definitions, the class labels assigned for each pair of generated token samples are shown in Table 3.5. Since the sentence Figure 3.5 consists of another token “5” with a “CD” POS tag, all generated token pairs will be assigned a label 0 since “5” does not fall within a gold quantity span.

Since I approached the MeasEval task by breaking down the entity spans into token pairs, I did not adhere to the official evaluation criteria (see AppendixB). Rather a modified approach was followed, where for each pair of tokens, I directly compare the gold label I assigned (discussed above) with the label predicted by the encoding schema.

3.3.4 Extracting Subtree encompassing target entities and representing features

After the token pairs (E_1, E_2) are generated for the three datasets, the next step is to extract the subtree from the dependency parse tree encompassing the pair of target entities. This is achieved by traversing the shortest path from E_1 to E_2 through the edges of the dependency tree. All nodes/tokens encountered during the traversal between the two target nodes are considered to be part of the subtree.

Adjacency matrix (A) representation: Once the subtree is extracted for each pair (E_1, E_2) , it is represented using the adjacency matrix A . Since the dependency parse tree has governor-dependent relationships and a dependent can only have one governor, the rows in the A matrix represent the dependents and the columns represent the governor. If a pair (E_1, E_2) has n nodes within the subgraph, the A matrix will have dimension $n \times n$ (that is, $A \in \mathbb{R}^{n \times n}$).

Information matrix (IM) representation: The IM matrix is constructed to contain feature vectors of each node / token within the extracted subtree between the target entities (E_1, E_2) . Each row in IM corresponds to each row in the A matrix. Dimension $IM \in \mathbb{R}^{n \times c}$, where n represents the number of nodes in the subtree and c represents the size of **feature vector** of a given node. The feature vector here is a concatenation of the linguistic features of each node in the subtree. These linguistic features were obtained from the GATE preprocessing pipeline and experimented with different representation techniques, as the features are categorical. Since experiments of different encoding schemas were performed on all three datasets, basic and generic linguistic characteristics were chosen for IM rather than task-specific features. However, it should be emphasized that the IM constructed is not restricted to these features and should be explored in the future with more task-specific features related to the task. The features chosen to represent in the IM in this thesis are:

- POS tags of tokens
- Dependency relationship
- Token embedding
- Distance Labeling

To individually encode these features as feature vectors, multiple representation techniques were attempted to assess the impact of each technique. The following is the list of techniques applied for each feature representation.

POS tag: The Stanford Parser (POS tagger) (Klein & Manning, 2003) module used in the GATE pipeline uses the Penn Treebank tag set (Marcus et al., 1993), which consists of 36 POS tags and other tags for punctuation and symbols. The number of unique punctuation marks and symbols observed in the training set varied according to the data set. To transform the POS tag into a numerical representation, two techniques were explored.

- (1) *Label encoding:* In this technique, all unique POS values observed in the training data were collected, sorted alphabetically (to maintain consistency), and assigned a label

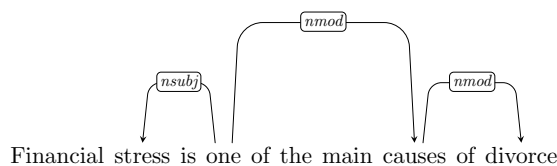
to each POS tag corresponding to the index of the sorted list as its label. During the construction of the IM matrix, each node/token in the subtree is assigned a single numerical value as a label representing the POS tag of the node/token.

- (2) *One-hot-encoding*: Similar to the label encoding technique, all unique POS values observed in the training data were collected, alphabetically sorted (for consistency) and stored as a POS reference list. During the construction of the IM matrix, for each node/token in the subtree, a vector of the same size as the POS reference list is created and assigned a value 1 to the index corresponding to the POS tag of the token in the reference list while filling the remaining elements of the vector with 0s.

Dependency relationship: The Stanford Dependency parser module (de Marneffe et al., 2006) was used in GATE to obtain the dependency parse tree of each sentence. Although the A matrix represents the structure of the subtree (that is, which node / token is connected to which other node/token), it does not encode the information of the dependency relationship (that is, the type of edge). Therefore, two approaches were experimented to encode the dependency relationship between the nodes in IM :

- (1) *Representing the immediate governor*: This was an initial step in trying to encode the dependency relationship. Since a node in the dependency parse tree can only have a single parent, the A matrix denotes who the parent is, but not the type. Therefore, to encode the type of dependency relationship of a token with its immediate governor(parent), the **One-hot-encoding** technique was used. Similarly to POS one-hot encoding, all dependency edge relationships observed in the training data were collected, sorted alphabetically, and stored as a dependency reference list. During the construction of IM , for each node/token in the subtree, a vector of the size of the dependency reference list is created and filled with 0s, except for the index of the vector corresponding to the dependency relationship of the node with its parent. It is marked as 1.
- (2) **Representing path of each node from the base node**: For MeasEval task, ME and MP are determined based on the relationship the entities have with Q. Therefore, instead of encoding the immediate edge relationship explained above, a path representation from E_1 to E_2 was chosen to represent it in IM . For MeasEval, E_1 always corresponds to *base_CD* and all paths were retrieved with a correspondence to it. Although this is not the case for the other two datasets (DDI and Sem10), to maintain a consistent approach throughout all data sets, I decided to retrieve the path

representation for all nodes in the subtree starting from E_1 , which is the first target entity node that appears in the sentence ordering for DDI and Sem10.



$$\begin{aligned}
 p(stress, stress) &= (self) \\
 p(stress, one) &= (nsubj) \\
 p(stress, causes) &= (nsubj, nmod) \\
 p(stress, divorce) &= (nsubj\ nmod\ nmod)
 \end{aligned}$$

Figure 3.7: Dependency subtree of a Sem10 sample, $(E_1, E_2) = (stress, divorce)$; Dependency path $p(E_1, t_i)$ generated for all nodes / tokens within the subtree with respect to E_1

As shown in Figure 3.7, dependency graphs are directed. However, to traverse along the edges to obtain a path representation (only), the dependency parse tree was treated as undirected³. Therefore, regardless of the direction of the arc in the dependency tree, the relationship between the dependent and the governor is taken as is, as shown in Figure 3.7. In addition, to indicate a path from a token to itself, a *self* path was also created rather than not encoding the path information at all. Once the path information is retrieved from the subtree, I compare two encodings:

- *Multi hot encoding*: A multi-hot encoding is typically used to represent a single categorical variable with a binarized label (see Section 2.4.3). However, a modified implementation of multi-hot encoding was utilized to represent the path information. Instead of binarizing the label of a dependency relationship, similar to one-hot encoding, the multi-hot encoding encodes all the edges encountered during the traversal from E_1 to a specific node in the dependency subtree, by adding 1 in the vector for each type of dependency relationship encountered. Hence the multi-hot path vector will be of size same as the dependency reference list, with multiple counts denoting the dependency relationship types encountered and the rest of the elements in the vector filled with 0s as shown in Figure 3.8

³Contrasting to this approach, another approach was attempted by introducing reverse relationship in the dependency graph with prefixed “r” (e.g., rnsbj, rnmob). However, after experimentation, it was decided to treat the graph as undirected to obtain path information rather than introduce more variables

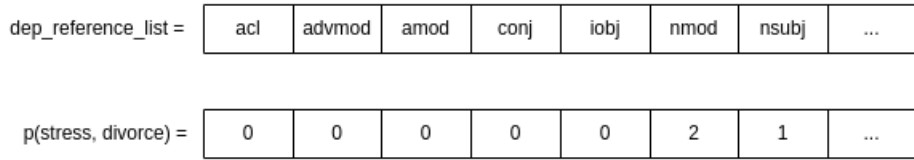


Figure 3.8: Multi-hot encoding of dependency path of $p(\textit{stress}, \textit{divorce}) = (\textit{nsubj} \textit{nmod} \textit{nmod})$

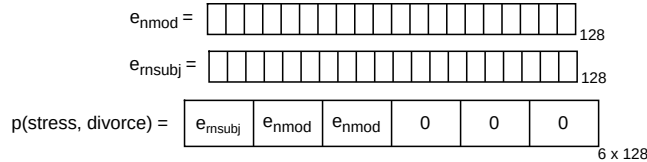


Figure 3.9: Concatenated node2vec encoding of dependency path $p(\textit{stress}, \textit{divorce})$; $lp = 6$ and $de = 128$

Although multi-hot encoding encodes all the dependency relationship types encountered during the traversal, the sequence of the path is not retained, i.e., all the dependency relationship types observed in the path are encoded without a particular order of the edges. Therefore, the following technique was attempted as a remedy for this problem.

- **Concatenating Node2Vec embedding:** This approach was used to encode dependency relationships with dependency embeddings (similar to Word2Vec embeddings). A modified Node2Vec approach (Grover & Leskovec, 2016) (see Section 3.3.4.1) was used to create dependency embeddings of size 128, for each type of dependency relationship observed in the training data. In IM , a vector of size $v \in R^d$ is instantiated, where $d = lp \times de$ where lp is the size of the maximum path length observed in the training data, and de is the size of the dependency embedding (in our experiment it is of size 128). For each row in IM , the representation of the path is constructed by concatenating the dependency embedding of each type of dependency relationship observed on the path. For paths shorter than the observed maximum path length (lp), 0s are padded until the vector size reaches the empirical limit d , as shown in Figure 3.9.
- **Averaging Node2Vec embedding:** Similar to the concatenation of Node2Vec embeddings, in this approach, the dependency embeddings were summed for all dependency relationships observed in the path. To normalize the vector, an average was calculated by dividing by the path length. Therefore, the averaged path representation was equal to the size of the dependency embedding, 128.

Token Embedding: Tokens were represented in *IM* with word embeddings retrieved from pre-trained ELMo models (Peters et al., 2018):

- *Original ELMo embedding:* Original ELMo model trained on 800M tokens from the WMT news crawl data was used to obtain the word embedding of 1024 size to encode tokens in *IM*.
- *PubMed ELMo embedding:* The PubMed ELMo model, trained with 10M PubMed abstracts, was used to obtain word embeddings of 1024 sizes to encode in *IM*.

Distance labeling: Inspired by (M. Zhang & Chen, 2018), the Double Radius Node Label (DRNL) approach was used to calculate the combined distance of a node v_i in a subtree to the pairs of target entities (E_1, E_2). Then, this label was represented as a one-hot vector in *IM*. The objective of this feature is to give a notion of the distance of how far a node is from the target pair of tokens. Both target nodes E_1 and E_2 carry the label 1. For the rest of the nodes v within the subtree, the distance label is calculated using the following hashing function:

$$f(v) = 1 + \min(d_{t_1}, d_{t_2}) + \lfloor \frac{d}{2} \rfloor \times (\lfloor \frac{d}{2} \rfloor + [d\%2] - 1) \quad (3.3)$$

where $f(v)$ assigns labels to all nodes v , d_{t_1} , and d_{t_2} are distances of v with respect to t_1 and t_2 , respectively. $d = d_{t_1} + d_{t_2}$, $\lfloor d/2 \rfloor$ is the integer quotient and $[d\%2]$ is the remainder of d divided by 2. Based on Figure 3.7, $f(stress)$ and $f(divorce)$ will have a distance label of 1, $f(love)$ ($d_{t_1} = 1, d_{t_2} = 2$) and $f(causes)$ ($d_{t_1} = 2, d_{t_2} = 1$) will have a distance label 3 based on the above hash function.

3.3.4.1 Node2vec

As discussed above, the dependency path is a feature represented in *IM*. The dependency path refers to the sequence of dependency relationships that exist when traversing from one token to another along the dependency graph. To represent the dependency relationship, dependency embeddings were created using the **Node2Vec** technique with a minor modification.

Node2Vec (Grover & Leskovec, 2016) is an embedding technique used to transform graphs into numerical representations. More precisely, it creates an embedding for each vertex in a graph, preserving the structure of the original graph so that similar vertices in the graph will have similar representations. This objective is similar to the concepts of Word2Vec (Mikolov et al., 2013) (see Section 2.4). Therefore, Node2Vec uses the same technique as Word2Vec(skip-gram model) to obtain a node representation. Since embedding

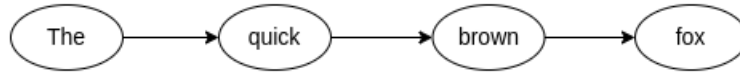


Figure 3.10: Directed acyclic graph with a maximum degree of 1.

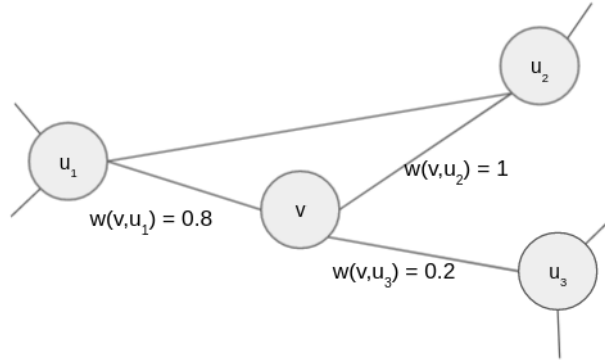


Figure 3.11: First-order transition probabilities

is created using the Word2Vec approach, Node2Vec focuses mainly on **sampling strategy** to create a corpus from the graph data to train the Word2Vec skipgram model.

Before looking at the sampling strategy, let us first understand the structure of the input used to train Word2Vec. The training data used in Word2Vec are made up of a sequence of tokens. As shown in Figure 3.10, this sequence can be interpreted as a directed acyclic graph with a maximum degree of 1. Since Word2Vec can embed this very specific kind of graph, (Grover & Leskovec, 2016) proposes a hyperparameterized sampling strategy based on **second-order random walk**. This strategy extracts the specific kind of graph structure as above, from more complex graph structures which can be (un)directed, (un)weighted, (a)cyclic, and with many degrees of connection.

A **first-order random walk** can be thought of as a walk carried out on a graph along the edges of the graph. At each step, the decision on where to go next is made on the basis of the current state alone. Figure 3.11 shows an example of a first-order random walk, where u_1, u_2, u_3 are three neighboring nodes of node v with weights w associated with each edge. Given these weights, the probability of picking the next step can be calculated as follows.

$$p(u|v) = \frac{w(u, v)}{\sum_{u \in N_v} w(u, v)} = \frac{w(u, v)}{d(v)} \quad (3.4)$$

where N_v is the set of neighboring nodes of v which are u_1, u_2 and u_3 in the given example and $d(v)$ denotes the degree of node v .

Based on Equation 3.4 the transition probabilities from v to u_1, u_2 and u_3 are 0.4, 0.5, and 0.1, respectively. Based on the transition probability, the next step from node v would

be toward u_2 and this is called *1-hop transition*. Therefore, a first-order random walk can be described as a walk generated by performing multiple 1-hop transitions, which only depends on the current state. However, **second-order random walk** proposed by [Grover and Leskovec \(2016\)](#) is a slightly modified version of the first-order random walk, which incorporates information from the previous step before deciding the next step during the traversal. This strategy consists of 4 arguments:

- **Number of walks (N):** Number of walks to be generated from each node in the graph
- **Walk length (L):** Maximum number of nodes in each walk
- **P:** Return hyperparameter
- **Q:** In-out hyperparameter

This means that the second-order random walk will go over each node in the graph, generating N number of walks, of length L . However, to decide which node to traverse next given the current state, the algorithm uses two parameters \mathbf{P} and \mathbf{Q} . Given the example shown in [Figure 3.12](#), let us assume that we have transitioned from node u_2 to node v . Therefore, the parameter \mathbf{P} is considered as the probability of going back from node v to node u_2 and the parameter \mathbf{Q} is considered as the probability of exploring the undiscovered parts of the graph. Based on these parameter values, the second-order transition calculates the normalized transition probability as follows.

$$p(u|v, t) = \frac{\alpha_{pq}(t, v)w(u, v)}{\sum_{u \in N_v} \alpha_{pq}(t, u)w(u, v)} \quad (3.5)$$

where t refers to the previous state (in our example ([Figure 3.12](#)), u_2 corresponds to t) and α is calculated as follows:

$$\alpha_{pq}(t, u) = \begin{cases} \frac{1}{p} & \text{if } d_{tu} = 0 \\ 1 & \text{if } d_{tu} = 1 \\ \frac{1}{q} & \text{if } d_{tu} = 2 \end{cases} \quad (3.6)$$

where d_{tu} denotes the shortest path distance between nodes t and u . Since d_{tu} must be one of 0,1,2, [Grover and Leskovec \(2016\)](#) conclude saying that only two parameters \mathbf{P} and \mathbf{Q} are sufficient to guide the walk by controlling how fast the walk explores and leaves the neighborhood of a particular node on the graph. Therefore, by decreasing the value of q we increase the probability of going outward (that is, not restricting the walk to a localized

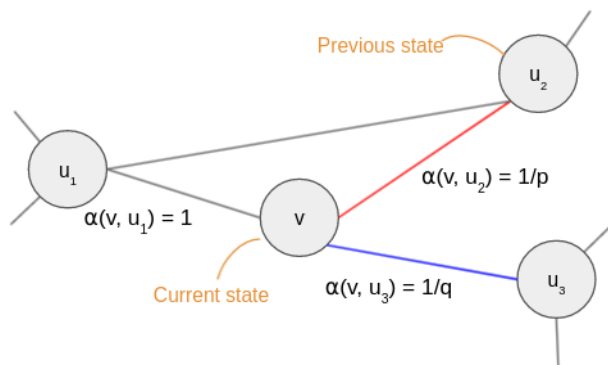


Figure 3.12: Illustration of random walk procedure in node2vec. The walk just transitioned from u_2 to v and is now evaluating its next step out from v . Edge label indicates search biases α ; yellow line indicates the in-out edge, and red line indicates the return edge.

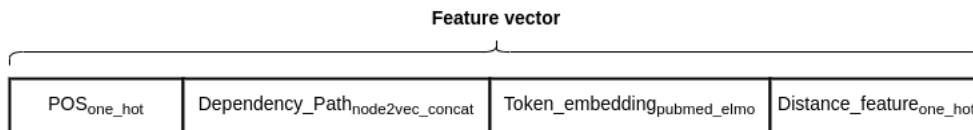


Figure 3.13: Detailed view of the feature vector in *IM*

neighborhood) and, conversely, restricting the walk within a localized neighborhood by increasing the value of q .

As mentioned above, Node2Vec uses the sampling strategy \mathbf{N} times on each node in the graph, producing a corpus of node sequences. This corpus is then fed into the Word2Vec skip-gram algorithm to generate node embeddings.

Since our main objective of using Node2Vec is to generate embeddings for dependency relationship types, instead of generating a sequence of nodes, I modified the Node2Vec sampling algorithm to generate a sequence of edges (i.e., a sequence of dependency relationship types) as the corpus to train the Word2Vec model.

3.3.5 Final representation of *IM*

Section 3.3.4, describes the different techniques used to represent the linguistic features (POS tag, dependency relationship, and token embedding) and the distance feature. Based on a preliminary ablation study conducted on the MeasEval data set with DGCNN architecture (see Appendix A), certain representation techniques were chosen for each feature as shown in Table 3.6.

Using these representation techniques, the linguistic features are transformed into vector representation and concatenated with each other (as shown in Figure 3.13) to construct the

Table 3.6: Representation techniques used to represent linguistic features in *IM*; bold text indicate the chosen representation technique for each feature.

Features	Representation techniques
POS tag	Label encoding One-hot encoding
Dependency relationship	One-hot encoding Label encoding Multi-hot encoding Node2Vec average Node2Vec concatenation
Token Embedding	Original ELMo PubMed ELMo
Distance feature	One-hot encoding

feature vector of each node / token in the *IM*. Although different representation techniques were explored, they were chosen on the basis of the MeasEval data set and used to represent the other two data sets to evaluate the impact of linguistic features on different encoding schemas (see Chapter 4). However, different features and their representation techniques may behave differently on different data sets. Due to limited processing resources and time constraints, a complete ablation of these features and their transformation techniques was not performed on all data sets.

Chapter 4

Experiments and Analysis

Previous chapters covered the motivation behind using a dependency parse tree as a base structure for the relation extraction task along with details of Graph Neural Networks (GNNs) that are designed to work on graph based data structures. Even though the idea of representing input in the form of an Information Matrix (IM) originated from GNN based architectures, it is not restricted to GNNs only. In this chapter, I evaluate the performance of a GNN based architecture DGCNN, along with other neural encoding models that are widely used in NLP research using IM as the primary or complimentary input. These models can be basically grouped into:

- (1) GNN based architecture: DGCNN
- (2) Classic baseline algorithms
- (3) Pretrained model - BERT
- (4) Ad hoc Transformer - Encoder stack

For evaluating and comparing the performance of the encoding systems, I used macro-averaged precision, recall and F1 measure for all the systems. Prior to discussing about the different experiments conducted, let me briefly discuss the evaluation metrics in the section that follows.

4.1 Evaluation metrics

As mentioned before in this thesis, I have treated the Relationship Extraction (RE) task as a classification problem. Where for each entity pair (E_1, E_2) a class label is predicted indicating the relationship between the the entities. This label is directly compared with the

gold label either provided by the organizers (for DDI, and Sem10) or derived (for MeasEval, see Section 3.3.3) to evaluate the systems' performance. For this purpose, **Precision (P)**, **Recall (R)** and **F1-measure (F1)** are used as the evaluation metrics. Out of the three data sets used, two of the data sets are treated as a multi-class classification problem (Sem10 and MeasEval), where MeasEval was treated as two separate multi-class classification tasks to detect ME and MP relationship individually¹. Hence, more precisely **Macro averaged** precision, recall and F1-measure were used to evaluate and compare performance of different encoding systems.

In a multiclass setting, macro average reduces the multiclass prediction into multiple sets of binary prediction and calculates each metric, precision, recall and F1-measure for each binary cases and then averages the results together (an example calculation is shown with respect to precision). Since the average is obtained by dividing by the number of classes seen in the data set, implicitly it is assumed that the each class gets equal weight when contributing their portion during the calculation of any evaluation metric.

Precision: also known as *positive predictive value*, is the fraction of relevant instances among the retrieved instances. It can be measured by the following equation:

$$P = \frac{TP}{TP + FP} \quad (4.1)$$

However to calculate the macro averaged precision, the following equation is used.

$$P_{macro} = \frac{P_1 + P_2 + \dots + P_k}{k} = P_1 \frac{1}{k} + P_2 \frac{1}{k} + \dots + P_k \frac{1}{k} \quad (4.2)$$

Recall: also known as *sensitivity* is the fraction of retrieved relevant information. It is calculated as follows:

$$R = \frac{TP}{TP + FN} \quad (4.3)$$

In the above equations (4.1, 4.3) TP, FP and FN stands for *True Positives*, *False Positives* and *False Negatives* respectively. **TP** refers to positive instances that are correctly identified by the system, while **FP** refers to instances that are falsely identified as positive instances and **FN** refers to positive instances left unidentified by the system.

¹Entities in MeasEval are inter-dependant on one another. Based on early experimentation, detection of ME and MP separately showed comparatively better performance than trying to predict both in the same system. Therefore all experimentation in this chapter follow the same process as to predict the two entities separately, in a multi-class setup

F1-measure: provides a single measure to represent precision and recall. In general, it is calculated as follows:

$$F_{\alpha} = (1 + \alpha^2) \times \frac{\textit{precision} \times \textit{recall}}{\alpha^2 \times \textit{precision} + \textit{recall}} \quad (4.4)$$

where α denotes the importance of recall over precision. However, to strike an equal balance between precision and recall, F1 (also known as *harmonic mean*) is calculated as follows:

$$F1 = 2 \times \frac{\textit{Precision} \times \textit{Recall}}{\textit{Precision} + \textit{Recall}} \quad (4.5)$$

4.2 GNN based architecture: DGCNN

The input representation in the form of Information matrix (*IM*) was inspired by the Graph Neural Networks (GNN). Since the inspiration of the GNN architecture was obtained from [M. Zhang and Chen \(2018\)](#), I used their DGCNN ([M. Zhang et al., 2018](#)) (see Section 3.1.1) on my relation extraction task.

4.2.1 Original DGCNN (DGCNN_{org})

As mentioned above an initial set of experiments of the DGCNN system is conducted on the original setup of the authors of the SEAL architecture, [M. Zhang and Chen \(2018\)](#). The authors used this architecture for link prediction between two nodes given a subgraph. As explained in Section 3.1.1, the DGCNN system consists of four sequential stages, the *graph convolution layers*, *sort pooling layers*, *1D convolution network* and *dense layer*. Table 4.1 shows the default parameter setting at each stage in the DGCNN architecture defined by the authors. The DGCNN system was trained for 6 epochs with an Adam optimizer and learning rate $1e - 4$.

4.2.2 Modified System (DGCNN_{mod})

As for parameter modification I experimented with changing the number of graph convolution layers and size of each layer, while keeping the rest of the parameters fixed as shown in Table 4.1.

- Graph convolution layers
 - Number of graph convolutions: 10
 - Window size at each layer: [4000, 2500, 1024, 512, 256, 108, 64, 32, 16, 1]

Table 4.1: Default parameter setting of each stage in the DGCNN architecture defined by the authors (M. Zhang & Chen, 2018)

Stage	Parameter	Values
Graph convolution layers	# of Graph convolutions	4
	Window size at each layer	[32, 32, 32, 1]
Sort pooling layer	Padding/Truncating size (k)	10
1D convolution network	First convolution layer	kernel size: $k(\sum_1^h c_t)$ Stride size: $k(\sum_1^h c_t)$ Activation function: ReLU In channel: 1 Out channel: 16
	Max pool layer	Kernel size: 2 Stride size: 2
	Second convolution layer	Kernel size: 5 Stride size: 1 Activation function: ReLU In channel: 16 Out channel: 32
Dense layer	Hidden layer	128
	Activation function	ReLU
	Loss function	CrossEntropyLoss

My main intention to modify only the graph convolution layer values is to retain original information from IM as much as possible. This is because, according to the DGCNN implementation, values derived from graph convolution layers are concatenated and used as input to the proceeding stages of DGCNN while discarding the original IM matrix (see Section 3.1.1.1). Since IM with token feature vectors are richer unlike node feature vectors derived from graph data (used as input in the author’s paper (M. Zhang & Chen, 2018)), I experimented by modifying the values of graph convolution layers in such a way that it would extract more information from the IM .

Table 4.2: Precision(P), Recall(R) and F1-measure(F1) results on all 3 data sets with original DGCNN ($DGCNN_{org}$) and parameter modified DGCNN ($DGCNN_{mod}$).

Experiment	Sem10			DDI			MeasEval (MP)			MeasEval (ME)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
$DGCNN_{org}$.56	.62	.56	.76	.87	.80	.58	.72	.61	.58	.72	.61
$DGCNN_{mod}$.68	.73	.69	.81	.88	.84	.60	.72	.63	.62	.73	.65

Table 4.2 shows increase in performance for $DGCNN_{mod}$ across all the data sets compared to $DGCNN_{org}$. This is because the graph convolution extracts multi scaled substructure features by performing graph convolution as shown below in equation 4.6:

$$Z^{t+1} = f(\tilde{D}^{-1}\tilde{A}Z^tW^t) \tag{4.6}$$

where Z^{t+1} is the output after t number of graph convolution cycles. $Z^0 = IM$, and W^t is the weight matrix of the t^{th} convolution layer (see Section 3.1.1.1 for complete detail on graph convolution). At each cycle of the graph convolution, the W (weight) matrix performs a linear transformation on the input Z from the previous layer. Say at the first convolution cycle, $Z^0 = IM \in \mathbb{R}^{n \times c}$, and $W \in \mathbb{R}^{c \times c'}$. Therefore, the size of the output from the 1st convolution layer $Z^1 \in \mathbb{R}^{n \times c'}$, which will then be used as input to the next layer. The final output of the graph convolution layer in DGCNN (which is then used as input in the SortPooling layer) is the concatenation of $Z^{1:t+1}$, excluding Z^0 which is the original IM .

The authors of DGCNN (M. Zhang & Chen, 2018) defined a size [32, 32, 32, 1] indicating that, 4 graph convolution cycles will take place where at each cycle the size of W matrix (more precisely the second dimension of the W matrix) is equal to the size defined in the list. In other words, at the first cycle $W^0 \in \mathbb{R}^{c \times 32}$ (where c is the size of feature vector in $Z^0 = IM$ and 32 is the size given as the parameter in the list). Even though the size defined by the authors M. Zhang and Chen (2018), is more suitable for the graph data used in their experiments², as they represent each node in the graph with an embedding size of 128 (i.e., $c = 128$), it is not suitable for my experimentation as the my node representations in IM is more richer in features (e.g. $c_{DDI} = 3800$). Therefore, using a weight matrix at the first graph convolution cycle with a size $W^0 \in \mathbb{R}^{3800 \times 32}$ on $IM \in \mathbb{R}^{n \times 3800}$ (along with other matrix multiplication as in equation 4.6) will result $Z^1 \in \mathbb{R}^{n \times 32}$, abstracting from 3800 features to 32 features. This results in huge feature loss. Therefore, the experimentation with sizes similar to the IM and gradually decreasing them performs comparatively better.

4.3 Encoding Information matrix

Information matrix used as input for DGCNN is a two-dimensional matrix where the rows represent the number of nodes contained within a subgraph and the columns correspond to the feature vector (i.e., a concatenation of multiple embeddings representing selected features) of each node. Even though DGCNN is configured in a way to accept the input (IM) as a two-dimensional matrix, for certain models such as feed-forward network or random forest networks (which were used as baseline models which will be discussed in detail in the following section), this increases the level of complexity of handling the data. Therefore, the transformation of two-dimensional IM into one dimensional vector was experimented with two approaches.

²PROTEINS (Borgwardt et al., 2005) or MUTAG (Debnath, Lopez de Compadre, Debnath, Shusterman, & Hansch, 1991) were few data sets which were used in the paper to evaluate the performance of the DGCNN

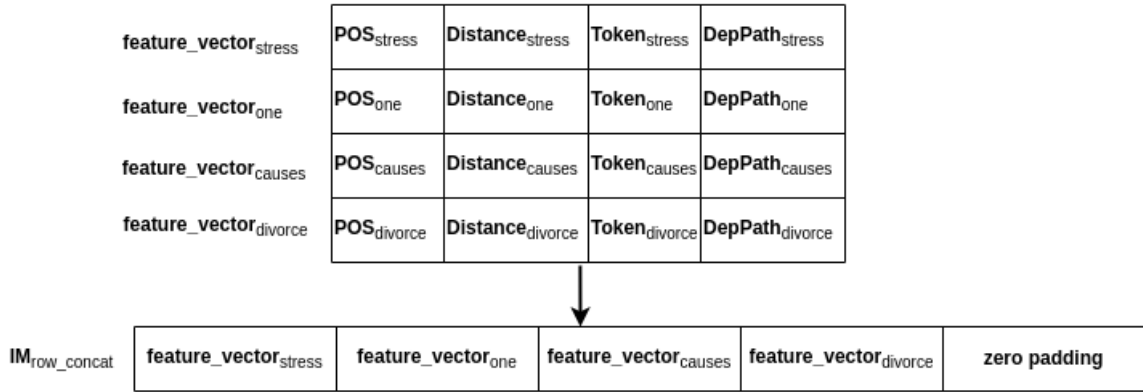


Figure 4.1: Row concatenation approach to construct one-dimensional vector

- (1) **Concatenating the rows (row_concat)**: In this approach, each row in the IM is concatenated one after the other forming a one dimensional vector without any loss in information. However, due to varying size of the subgraphs for different samples, the one-dimensional vector after row concatenation will differ in size between samples. Therefore to construct uniform sized vector representation for all samples, during the formulation of this representation, the possible maximum number of nodes that can exist within a subgraph is determined and used to calculate the size of vector to be padded with zeros after the concatenation of rows (see figure 4.1). For example, let's assume the maximum number of nodes observed within a subgraph is of size 8. But the current processing subgraph only has 5 nodes and the size of the feature vector is 1000. Therefore, after concatenating 5 rows, we obtain a vector of size 5000. Since the maximum nodes can be 8, a remaining $3 * 1000$ vector with 0's is padded at the end of 5000 to normalize the size to 8000.
- (2) **Aggregating columns (col_aggr)**: In contrast to above, in this approach all rows in the IM are summed column-wise to obtain a lossy vector representation for each sample data (see figure 4.2). No padding was required in this approach since the size of the resulting vector is equal to the size of the feature-vector which is a fixed size for all samples in the corpus.

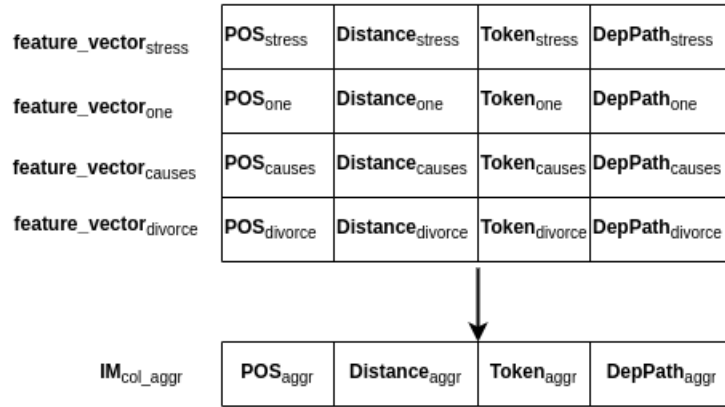


Figure 4.2: Column aggregation approach to construct one-dimensional vector

4.4 Classic baseline algorithms

Objective: The Graph Neural Network (GNN) inspired Information Matrix (IM) contains feature vectors describing all the nodes within a subtree. To contrast with complex architecture like DGCNN used above, in the following set of experiments I wanted to observe how simple baseline modules like Random Forest, Support vector machines or a Feed-Forward network would perform with IM input.

4.4.1 Random Forest (RF)

A random forest fits a number of decision tree classifiers on various sub samples of the data set and averages over the results of each decision tree classifier to improve the accuracy of the prediction while having control over preventing over-fitting. An off-the-shelf RF implementation from the sklearn library³ was used with the default parameter setting set to, `n_estimator = 100` and branching criterion (impurity measure) set to “*Gini*”. Input to RF was provided by transforming the IM with either of the above mentioned transformation technique (row concatenation or column aggregation). Therefore, the size of the input varied depending on the size of vector after the transformation of either approach and the output depends on the task at hand.

³<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

4.4.2 Support Vector Machines (SVM)

Similar to RF network explained above the off-the-shelf support vector machine (SVM) implementation provided by the sklearn library⁴ was used with the default parameter settings set to kernel being “*RBF*”, decision function shape set to “*ovr*” due to multi-class setting and the tolerance as the stopping criterion set to $1e - 3$. SVM was experimented with both transformed input variation of the IM matrix (row concatenation and column aggregation).

4.4.3 Feed-Forward Neural network with Non linear activation (FF)

Similar to both experiments above off-the-shelf feed-forward network⁵, also known as Multi-Layer Perceptron (MLP) implementation, with one hidden layer was used to experiment with *IM* as input, transformed with either of the approaches mentioned above. As mentioned above the size of the input vector depends on the size of the resulting vector after the transformation of either approach, the number of hidden neurons were fixed to 100 with ReLU as the activation function and the number of output neurons varied depending on the task at hand (i.e., DDI data set had two neurons in the output layer, Sem10 data set had 10 and MeasEval data set had 4 neurons in the output layer). The network’s weight was optimized using Adam (Kingma & Ba, 2015), with a learning rate of 0.001. The network was trained for a maximum of 200 iterations or less until it converged. Since the vanilla neural network was chosen to establish a baseline for the experiments, no parameter tuning was conducted to choose the values mentioned above. Instead, all values set as default by the sklearn library⁶ for this network were utilized as it is.

Table 4.3 shows the results of the baseline models on all three data set with both *IM* transformation approaches. In comparison to different machine learning models, based on F1 score and Recall (R) the models can be ranked in (descending) order as FF, SVC and RF regardless of the transformation approach. However, the precision fluctuates within different models, data sets and transformation approach. Except for Sem10 data set, RF model has higher precision value than FF models. Also in comparison to transformation approach within the RF experiments RF_{row_concat} comparatively performs better than RF_{col_aggr} . This observation might be because of RF models are highly capable of extracting distinguishable features, that works as branching factors. Aggregating all the row values in the *IM* into

⁴<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

⁵https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

⁶https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

one vector as in column aggregation diffuses the information, which might be impacting the performance of RF_{col_aggr} .

Table 4.3: Precision(P), Recall(R) and F1-measure(F1) results on all 3 data sets with baseline models RF, SVC and FF with 2 different encoding of the IM (IM_{col_aggr} and IM_{row_concat})

#	Experiment	Sem10			DDI			MeasEval (MP)			MeasEval (ME)		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
1	RF_{col_aggr}	.66	.6	.61	.88	.67	.72	.67	.50	.53	.80	.51	.53
2	SVC_{col_aggr}	.75	.72	.73	.87	.76	.80	.60	.57	.59	.73	.58	.58
3	FF_{col_aggr}	.73	.73	.73	.83	.85	.84	.63	.65	.64	.63	.64	.63
4	RF_{row_concat}	.68	.61	.63	.88	.66	.71	.80	.51	.53	.78	.52	.55
5	SVC_{row_concat}	.73	.69	.70	.87	.73	.77	.76	.58	.59	.76	.59	.61
6	FF_{row_concat}	.74	.74	.74	.84	.86	.85	.70	.67	.68	.69	.69	.69

However in contrast to FF and RF, where the row concatenation approach performs better than column aggregation, for SVC based experiments the opposite works best except for MeasEval data set. Even though there is no clear evidence to this behaviour it can be assumed that, due to the nature of how SVC works (that is based on the notion of identifying a hyperplane that best divides a data set) when the input dimensionality is high, data-points are more scattered in the vector space, impeding the model’s decision (curse of dimensionality). Considering the size of the vector after both transformation approaches, vector resulting after column aggregation has a lesser size compared to the size of the row concatenation. Therefore, SVC_{col_aggr} comparatively performs better than the SVC_{row_concat} .

In conclusion, compared to all experiments in Table 4.3, FF_{row_concat} shows better performance. Therefore, I chose the row concatenation transformation approach and FF as a better contender to compare with future experiments.

4.5 Pretrained baseline model - BERT

Objective: Currently, deep pretrained language models such as BERT are widely used in many NLP tasks achieving state-of-the-art performance. In contrast to DGCNN and other neural models trained with IM matrix as input, the objective of this experiment is to experiment with BERT as a trending baseline to observe the impact of pretrained language model on relation extraction tasks.

For this experiment, the uncased base pretrained BERT model from Hugging Face library ⁷ was used with a feed-forward network as a classifier on the top for performing the

⁷<https://huggingface.co/bert-base-uncased>

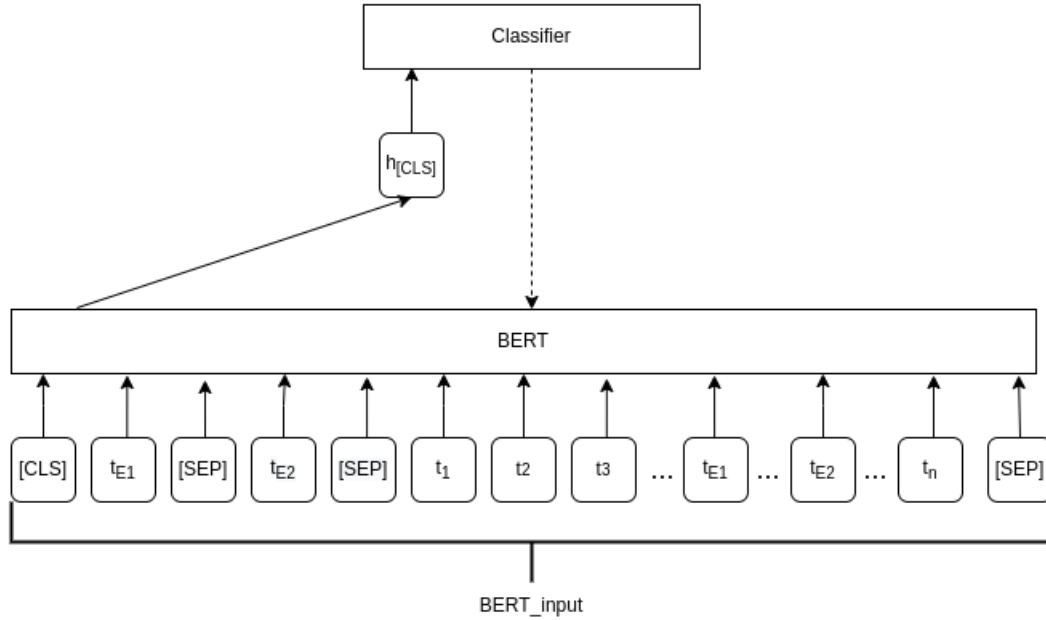


Figure 4.3: BERT language model combined with a FF network as classifier to perform RE task. Dotted lines indicate the back propagation path. BERT input constructed with 3 segments; E1, E2 and tokenized sentence, with [SEP] marker after every segment.

classification task. The FF is designed with 1 hidden layer with 50 neurons with ReLU as the activation function and CrossEntropyLoss⁸ as its loss function, with class weights calculated by $1/\text{total_num_datapoints_within_class}$. Loss calculated by the above function is used to adjust the weights of the FF while finetuning BERT weights according to the task. The model was trained for 2 epochs⁹, with AdamW optimizer and learning rate set to $2e - 5$.

Considering the nature of RE task, the input to BERT was constructed (Thillaisundaram & Togia, 2019) as depicted in figure 4.3. Where t_1 to t_n denotes the tokenized sentence with n number of tokens within a sentence. t_{E1} and t_{E2} denote the tokens that are identified as entities between which a relationship should be predicted. As pointed out by BERT authors (Devlin et al., 2019) special classification token [CLS] is added at the beginning of each input to obtain a sequence representation from BERT’s final layer to be used for the downstream classification task. Originally (Devlin et al., 2019) [SEP] marker is used to differentiate between different segments of the text (i.e., different sentences given a paragraph of text). However, I used the [SEP] marker as it was intended to be used (to distinguish sentences)

⁸<https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>

⁹Out of the 4 epochs trained, 2nd epoch on the development data set showed high performance

and to segment different parts of the BERT input. These different parts of the input are the entities E1 and E2 which are explicitly annotated for SEM10 and DDI data set and the derived ones for MeasEval data set. Therefore, the input to BERT is constructed with 3 segments, where each segment is split by a [SEP] marker. The first segment contains tokens annotated as E1, the second segment contains tokens annotated as E2, and the final segment is the entire tokenized sentence that contains both E1 and E2.

This representation puts the focus on the entities of interest, especially for the DDI data set and MeasEval data set, which contain multiple entities, with different relationships. Hence, using an input in the form of a tokenized sentence alone without explicitly annotating the entities will lead to confusion not knowing which entities to target to predict the relationship. Therefore, I opted to experiment with separating the entities outside the sentence with a [SEP] marker to distinguish target entity tokens from other tokens (Thillaisundaram & Togia, 2019). As for the classification part, the hidden state representation of the [CLS] token from the last layer is extracted and fed into the classifier to predict the relationship between the two target entities, explicitly annotated in the input to BERT. Moreover, this experiment was designed in a way to fine tune the BERT model by back propagating the error from the classifier to BERT parameters. Experiment results are given in Table 4.4 with other BERT based experiments discussed in the following section.

4.6 Concatenating Bert output with external representations

Objective: Even though BERT is trained in a self-supervised manner on a large corpus of texts obtaining state-of-the-art performance on many NLP related tasks, fusing external information utilizing varying techniques with regards to the task at hand has shown improvement in performance (A. Yang et al. (2019); Urooj, Mazaheri, Da vitoria lobo, and Shah (2020); Liu, Fu, Zhang, and Xiao (2021)). In the same stream of thought, the objective of the following experiments is to experiment fusing different encodings of IM with BERT’s representation to observe the impact on performance with regard to the RE task. Even though varying techniques were followed to fuse external information with BERT, I followed a simple concatenation of IM encoding with the final hidden state of the [CLS] token and used it as input to a FF network for classification¹⁰

4.6.1 Concatenating IM_{row_concat} as an external feature vector (BERT IM)

In this experiment, using the row concatenation approach (see section 4.3) the IM was transformed into a one-dimensional vector (IM_{row_concat}) to concatenate with the final

¹⁰The FF classifier is the same as the FF classifier discussed earlier for BERT model

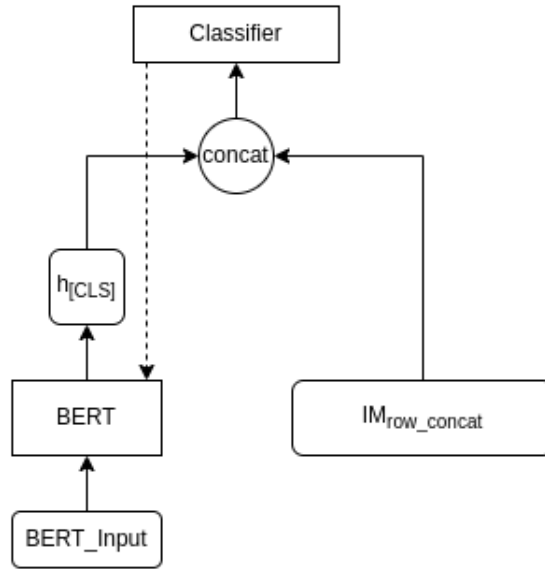


Figure 4.4: BERT_IM architecture; IM_{row_concat} concatenated with final hidden representation of [CLS] token from BERT as used as input to the FF classifier. Dotted lines indicate the back propagation path of the error.

hidden representation of the [CLS] token obtained from BERT and used as input to the classifier (see Figure 4.4). Since the IM matrix is a data representation of hand-crafted features, there are no weight parameters to fine-tune except for fine-tuning the BERT parameters based on the task. Therefore, as explained above (see Section 4.5), the error obtained during the classification task is backpropagated to adjust the weight parameters of the classifier as well as the BERT architecture during the training process.

4.6.2 Concatenating BERT and Convolved IM (BERT IM_{conv})

Objective: The objective of this experiment is to extract abstract features from the row concatenated IM (IM_{row_concat}) and use it as an external feature to concatenate with BERT’s [CLS] representation and observe the performance of the system on the RE task. The reason to perform a convolution on the IM_{row_concat} is because, as mentioned earlier, the IM is a data structure of hand-crafted features concatenated with one another to form a feature vector of each node in a subgraph, extracted from a dependency parse tree between 2 entities. The IM_{row_concat} is a simple concatenation of rows in the IM matrix to construct a single-dimensional vector that can be concatenated with another vector like the last hidden state representation of the [CLS] token. Similar to the BERT_IM experiment, here as shown in Figure 4.5 the vector obtained after performing the convolution on the IM_{row_concat} is

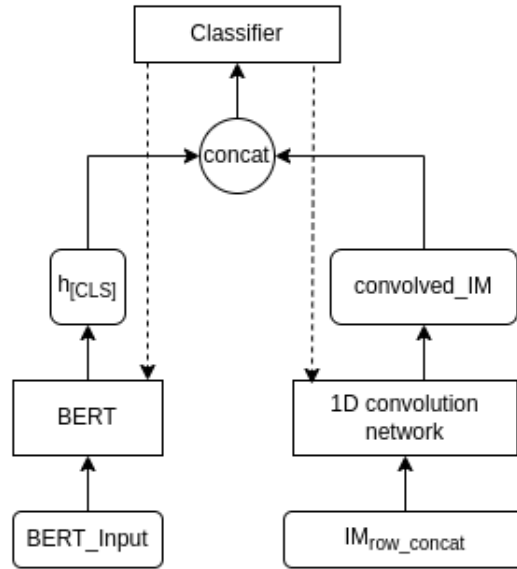


Figure 4.5: BERT- IM_{conv} architecture; concatenation of final hidden representation of [CLS] token from BERT with output of 1D convolution performed on IM_{row_concat} prior to feeding to a FF classifier. Dotted lines indicate the back propagation path of the error to fine-tune the weight parameters of convolution network and BERT.

concatenated with the final hidden state of the [CLS] token and used as input to a FF network for classification. The convolution network architecture used is same as the 1D convolution used in the DGCNN architecture (see Section 3.1.1.3). The error calculated after classification is backpropagated to adjust the weight parameters of the convolution network and fine-tune BERT.

4.6.3 Concatenating BERT and DGCNN (BERT-DGCNN)

Objective: The IM was constructed as an input to a Graph Neural Network (GNN), such as DGCNN. The DGCNN tries to learn important feature patterns from cascaded feature vectors (from neighboring nodes). Similar to the motive of above experiment (see Section 4.6.2) the objective of this experiment is to evaluate the effectiveness of concatenating the internal representation of DGCNN to the final hidden representation of [CLS] token of BERT. This concatenated vector is then used as input to a FF network for classification.

As shown in figure 4.6, the final hidden state of [CLS] token from BERT is concatenated with the feature vector derived as the output from the 1D convolution in the 3rd stage of the DGCNN system (i.e., the input to the dense layer in the DGCNN system). This concatenated vector is used as input to the FF network for classification. The error obtained

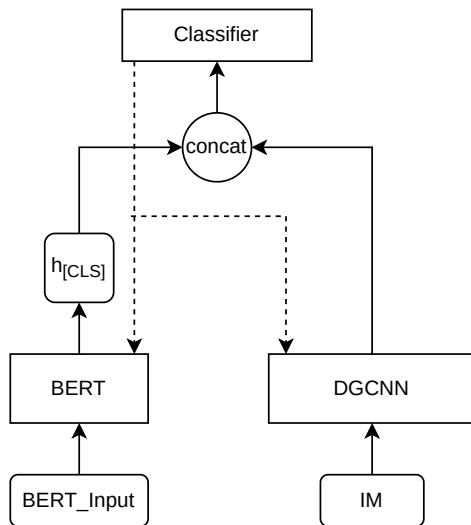


Figure 4.6: BERT_DGCNN architecture; concatenation of final hidden representation of [CLS] token from BERT with the intermediary representation obtained after the 1-D convolution stage of DGCNN, prior to using it as input to a FF classifier. Dotted lines indicate the back propagation path of the error to fine-tune the weight parameters of BERT and DGCNN.

during the classification task is backpropagated to adjust the weight parameters of both BERT and DGCNN architectures during the training process.

Table 4.4: Precision(P), Recall(R) and F1 measure(F1) results on all 3 data sets with BERT based architectures

Experiment	Sem10			DDI			MeasEval (MP)			MeasEval (ME)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
BERT	.83	.86	.84	.82	.85	.84	.71	.70	.70	.71	.71	.70
BERT_IM	.85	.86	.85	.85	.90	.87	.73	.73	.72	.72	.74	.72
BERT_IM _{conv}	.84	.86	.85	.81	.88	.84	.72	.73	.72	.72	.73	.72
BERT_DGCNN	.81	.81	.80	.81	.88	.84	.59	.72	.63	.62	.72	.65

Table 4.4 shows the macro averaged results of all BERT based architectures. Based on the numbers shown in the table, it can be observed that concatenating linguistic features in the form of IM_{row_concat} to BERT shows improvement on all data sets consistently compared to BERT alone results. Even though the recall value of BERT_IM and BERT are the same for Sem10 data set, combining IM information to BERT has improved the precision of BERT rather confounding. Therefore this improvement in performance across all data sets answers the hypothetical question of this thesis as to whether linguistic features are beneficial to transformer-based models like BERT in improving performance on the RE task. Additionally, this indicates the potential for more task specific features to improve

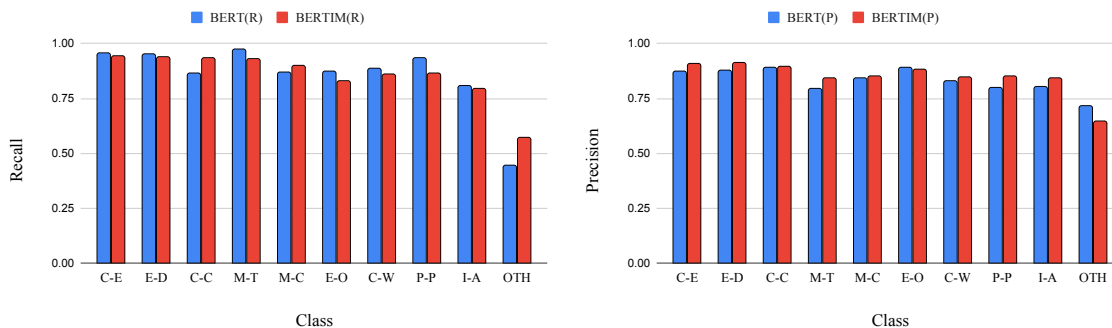


Figure 4.7: Recall (left) and Precision (right) values of BERT and BERT_IM model for each class in the Sem10 data set; OTH:Other, C-E: Cause-Effect, I-A: Instrument-Agency, P-P: Product-Producer, C-C: Content-Container, E-O: Entity-Origin, E-D: Entity-Destination, C-W: Component-Whole, M-C: Member-Collection, M-T: Message-Topic

the performance even further, as very basic linguistic features show improvement on the BERT model.

BERT-BERT_IM precision, recall comparison on Sem10 data set: Further breaking down the prediction results of BERT and BERT_IM, Figure 4.7 shows the recall and precision values of each class within the Sem10 data set. As the figure illustrates, BERT_IM has comparatively improved precision (right image) for all classes except for OTHER class than BERT alone. However, recall wise (left image), BERT model has comparatively high recall than BERT_IM except for CONTENT-CONTAINER (C-C) class and MEMBER-COLLECTION (M-C) class. Looking at the confusion matrix of both BERT (see Table 4.5) and BERT_IM (see Table 4.6), BERT confused C-C class more often (15 instances¹¹) with E-D (ENTITY-DESTINATION) class while there are only 2 instances¹² BERT_IM confused with E-D class. During error analysis, I was able to observe that some examples for both classes are much similar to one another, causing BERT to be confused with one another.

Ex 10: A tableau like this one is placed inside the king's room and refers to court ceremonies.

Gold class: (C-C), BERT_IM: (C-C), BERT: (E-D)

Ex 11: The instrument was sealed in a polyethylene bag for 40 hours at room temperature.

Gold class: (C-C), BERT_IM: (C-C), BERT: (E-D)

Ex 12: Then, the target PET bottle was put inside of a metal container, which was grounded.

Gold class: (E-D), BERT_IM: (E-D), BERT: (E-D)

¹¹out of the 15 examples, BERT_IM confused 2 instances with E-D class and 1 instance with OTH class

¹²BERT also got these 2 instances confused with E-D

Table 4.5: Confusion matrix of BERT on Sem10 data set

Gold Class	j Predicted class (BERT)										Total
	Oth	C-E	I-A	P-P	C-C	E-O	E-D	C-W	M-C	M-T	
Oth	200	30	19	32	13	20	20	39	32	44	449
C-E	4	314	1	3		2				4	328
I-A	13		123	10			1	5			152
P-P	1	4	3	206		2	1			3	220
C-C	4				166	1	15	6			192
E-O	13	7		5		224	2	3	1	2	257
E-D	8				4		278	1		1	292
C-W	17		6	1	2			271	4	5	306
M-C	15	4	1	1		2		1	201	6	231
M-T	4	1			1				1	254	261
Grand Total	279	360	153	258	186	251	317	326	239	319	2688

Table 4.6: Confusion matrix of BERT_{IM} on Sem10 data set

Gold Class	Predicted class (BERT _{IM})										Total
	Oth	C-E	I-A	P-P	C-C	E-O	E-D	C-W	M-C	M-T	
Oth	256	18	13	21	12	21	18	31	28	31	449
C-E	11	309		1		4				3	328
I-A	21		121	5			1	3			152
P-P	18	4	2	190		2	2			2	220
C-C	10				179		2	1			192
E-O	25	6		5		213	2	2	1	2	257
E-D	12				4		274	1		1	292
C-W	19		8	1	4			263	6	4	306
M-C	14	1				2		3	208	3	231
M-T	10	1						6	1	243	261
Grand Total	396	340	144	223	200	242	300	310	244	289	2688

Ex 13: We poured the pulp into the tupperware tub filled with water.

Gold class: (E-D), BERT_IM: (E-D), BERT: (E-D)

Ex 10 and Ex 11 are examples of class C-C which BERT confused with E-D. When comparing Ex 10 and Ex 11 with Ex 12 and Ex 13 the structure of the sentences seems almost similar, leading BERT to be confused with E-D class. However, features in *IM* (BERT_IM), helped BERT to predict the classes correctly as pointed out earlier based on the confusion matrices. Even though there is no clear way to point out which feature/features in *IM* exactly contributed for this correction, one common observation was that, out of the 15 misclassified cases 7 instances had an indication for passive in the dependency path (nsubjpass) encoded in the *IM*. However, I was able to observe some E-D examples with passive sentence structure as shown in Ex 14 and Ex 15 leading for no clear conclusion as to what contributed for the change, either dependency relationships or the token embeddings obtained from ELMo. However, the overall observation based on these analysis is that the concatenation of *IM* with BERT seems more precision oriented and improved the performance of BERT slightly on Sem10 data set.

Ex 14: This song has been entered into the contest unofficially.

Gold class: (E-D), BERT_IM: (E-D), BERT: (E-D)

Ex 15: The body that was removed for an autopsy was placed in a lead coffin, which was put inside a wooden case.

Gold class: (E-D), BERT_IM: (E-D), BERT: (E-D)

BERT versus BERT_IM performance comparison on DDI data set: Unlike Sem10, based on Table 4.4, BERT_IM shows a 3% increase in F1 score and 5% increase in precision on DDI data set. During error analysis, I found that the usage of PubMed ELMo embedding in the *IM* which is pretrained on medical data, has provided the added advantage to BERT_IM model to have a performance increase. This is further evident based on the results discussed with Table 4.15 in Section 4.8.

Table 4.7: BERT confusion matrix on DDI

Gold	Predicted		
	0	1	Total
0	4898	325	5223
1	218	715	933
Total	5116	1040	6156

Table 4.8: BERT_IM confusion matrix on DDI

Gold	Predicted		
	0	1	Total
0	4940	283	5223
1	142	791	933
Total	5082	1074	6156

In addition to the PubMed ELMo embedding, the construction of *IM*, limiting only to the scope of target entities prevents unnecessary confusions for BERT, boosting it's

performance when concatenated with *IM*. In other words, *IM* is constructed by encoding linguistic features of all nodes between the two target entities extracted from the dependency parse tree (see Section 3.3.4). In contrast to *IM*, BERT input is encoded by explicitly annotating target entities at the beginning of the tokenized sentence while having a [SEP] marker to separate entities and tokenized sentence (see Section 4.5). Even though I attempt to highlight the target entities in BERT input, in instances such as Ex 16 and Ex 17 the same drug name occurs in multiple places within the same sentence, where (*theophylline*¹ and *DIFLUCAN*) has a positive relationship and (*DIFLUCAN* and *theophylline*²) has a negative relationship. This made me realize that the highlight of entities alone without positional information to BERT is not sufficient. Concatenation of *IM* to BERT (BERT_IM) has the added advantage of boundary being limited only between the target entities and not having the duplicated drug name within the IM¹³ leading to a better performance.

Ex 16: Careful monitoring of serum theophylline¹ concentrations in patients receiving DIFLUCAN and theophylline² is recommended.

Gold class: FALSE, BERT_IM: FALSE, BERT: TRUE

Ex 17: Rifampin¹: Rifampin² enhances the metabolism of concurrently administered DIFLUCAN.

Gold class: TRUE, BERT_IM: TRUE, BERT: FALSE

Comparison of BERT_IM_{conv} and BERT_DGCNN: Apart from BERT-BERT_IM comparison, surprisingly, considering the capability of convolution networks to extract multiple abstract features from raw information, I was expecting both BERT_IM_{conv} and BERT_DGCNN to have a high performance than BERT_IM. Based on Table 4.4, even though BERT_IM_{conv} results are marginally better than BERT (except for DDI precision) on all data sets, BERT_DGCNN performs the worst of them all, in fact the concatenation of both the complex architectures degrades the overall performance (F1) than BERT as shown in Table 4.9. Also, comparing the values between different models shown in the table, it can be seen that the scores are more leaning towards DGCNN_{mod} than of BERT, implying that the classifier in BERT_DGCNN (see Figure 4.6) is more biased to DGCNN based input than of BERT (except for Sem10 data set).

In summary, external features in the form of *IM* increased the performance of BERT model on all data sets consistently. Even though the increase in BERT_IM is marginal compared to BERT, the concatenation of *IM* did not confound the performance of BERT.

¹³there are very few exceptions where the same drug name as one of the entity name can be in the dependency path between the target entities

Table 4.9: Comparison results of BERT based and DGCNN based models

Experiment	Sem10			DDI			MeasEval (MP)			MeasEval (ME)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
BERT	.83	.86	.84	.82	.85	.84	.71	.70	.70	.71	.71	.70
BERT_DGCNN	.81	.81	.80	.81	.88	.84	.59	.72	.63	.62	.72	.65
DGCNN _{mod}	.68	.73	.69	.81	.88	.84	.60	.72	.63	.62	.73	.65

Additionally, I see that the declarative nature of IM (where all features are explicitly provided) rather than compiled information as in BERT_IM_{conv} or BERT_DGCNN shows comparatively better results for the 3 data sets I experimented with.

4.7 Six layer Transformer Encoder Stacks (6TS)

Objective: Off the shelf BERT implementation requires input in the form of sentences or tokenized sentences. Providing additional linguistic features such as POS tags or Dependency information or any other handcrafted feature is not straightforward. However, BERT’s architecture or more precisely the encoder stack architecture have shown great performance improvements in recent research. Therefore, the objective of the following set of experiments is to mimic BERT based architecture in the form of Transformer encoder stack but with *IM* as input to evaluate if self attention can benefit from linguistic features.

As shown in Figure 4.8, the architecture of the transformer encoder stack (with 6 layers) was constructed following the implementation of Vaswani et al. (2017). The raw information matrix (*IM*) was used as input to this architecture. However, since the size of *IM* (specifically the number of rows) varies depending on the subtree, a maximum number of rows was defined and *IM* with a smaller number of rows was padded with zeros until the defined size. The input size of 6TS depends on the size of the feature vector in *IM*. All token outputs from the final encoder are concatenated before being fed to the classifier to predict the relationship. The architecture of the classifier is similar to the architecture of the classifier used in BERT. Containing only one hidden layer with 50 neurons, using ReLU as the activation function, and loss calculated with CrossEntropyLoss with weights passed as parameters to bias the back-propagation values depending on the class. The weights of the classes are calculated by obtaining the inverse of the total number of sample points within a class in the training data ($1/total_num_datapoints_within_class$). The 6TS was trained for a maximum of 4¹⁴ with the AdamW optimizer and the learning rate set to $2e-5$.

Table 4.10 shows the results of two 6TS experiments in the three data sets, where one of

¹⁴out of which a particular epoch was chosen based on the performance of dev set

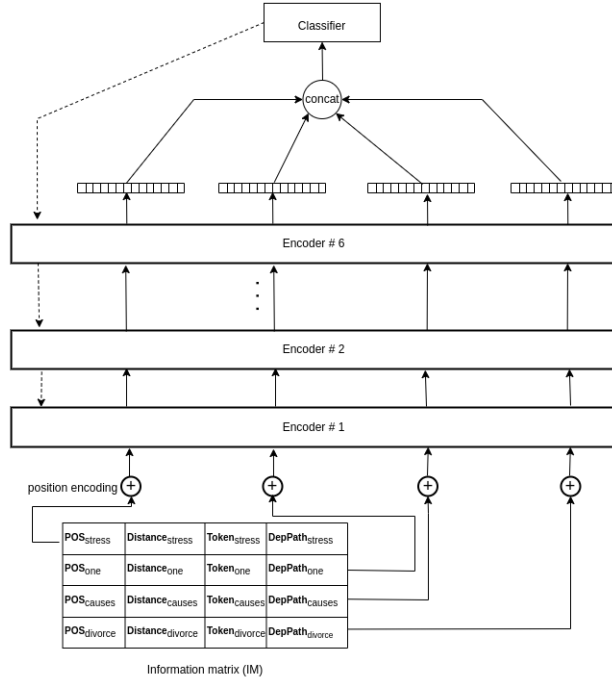


Figure 4.8: 6 layer Transformer encoder stack with *IM* as input

Table 4.10: Precision (P), Recall (R), and F1 measure (F1) on all 3 data sets based on 6TS. The ‘with’ and ‘no’ positions indicate whether positional encoding is embedded in the 6TS input.

Experiment	Sem10			DDI			MeasEval (MP)			MeasEval (ME)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
6TS _{with_position}	.75	.78	.76	.80	.83	.81	.73	.71	.72	.73	.72	.72
6TS _{no_position}	.75	.76	.76	.88	.84	.86	.73	.71	.72	.73	.73	.72

the experiments is with positional encoding and the other without positional encoding embedded with the input. According to the authors Vaswani et al. (2017) of the Transformer architecture, the purpose of embedding positional encoding in the input is to make the architecture aware of the ordering of tokens within a sentence, as the transformer architecture processes all tokens within a sentence at the same time, unlike the LSTM architecture (Hochreiter & Schmidhuber, 1997), where tokens are fed one after the other in a sequence. However, based on the results in Table 4.10, although the results of both experiments are very similar to each other, the experiment without positional encoding specifically for DDI shows a 5% increase in F1 measure from the experiment with positional encoding. This may be because the IM matrix that I use as input to the 6TS architecture is constructed on the basis of the subtree extracted from the dependency parse tree, which does not retain a sentence ordering of the tokens. Therefore, embedding positional encoding with each node in IM provides a false order to the 6TS architecture, which could hinder the performance of $6TS_{with_position}$.

4.8 Overall comparison

Table 4.11 shows a summary of the different encoding systems discussed above. The second best scores are highlighted with gray.

Table 4.11: Summary of the results of the best performing configuration of different encoding systems

Experiment	Sem10			DDI			MeasEval (MP)			MeasEval (ME)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
FF_{row_concat}	.74	.74	.74	.84	.86	.85	.70	.67	.68	.69	.69	.69
$DGCNN_{mod}$.68	.73	.69	.81	.88	.84	.60	.72	.63	.62	.73	.65
BERT	.83	.86	.84	.82	.85	.84	.71	.70	.70	.71	.71	.70
BERT-IM	.85	.86	.85	.85	.90	.87	.73	.73	.72	.72	.74	.72
$6TS_{no_position}$.75	.76	.76	.88	.84	.86	.73	.71	.72	.73	.73	.72

Difference in the nature of the data set: The main observation of Table 4.11 is that different data sets pattern differently. BERT-IM being the best contender comparative to all settings consistently shows a high macro-average F1 on all data sets. However, the configuration placed second varies depending on the data set. Taking into account the macro-average F1 score, BERT is second for Sem10, and for the other two data sets (DDI and MeasEval), $6TS_{no_position}$ is second. Furthermore, considering the range of variation in terms of the F1 score between different configurations, Sem10 varies with a spread of 16%,

DDI with 3%, and MeasEval approximately with 7-9%. This shows the difference in the type of relation tasks addressed by different data sets along with the nature of the data set. The Sem10 data set is composed of general English sentences, where entities are common English words. As shown in example Ex 18, Ex 19 and Ex 20, certain relationships are predicted based on the real meaning and real world relationship of the entities, rather than looking at the context of those entities alone. In other words, based on example Ex 19, a classification model is expected to predict that *posts* (product) are produced by a *writer* (producer), when the context does not explicitly state it. Since BERT is trained with a large corpus of texts (Wikipedia-2500 million words and book corpus 800 million words), BERT has these associations to associate *posts* are being written by *writer*, providing BERT an added advantage over the other encoding systems.

Ex 18: And you thought a dentist with a drill was scary.

Gold class: INSTRUMENT-AGENCY (I-A)

Ex 19: When viewers ignore negative posts, the offending writer often leaves the site.

Gold class: PRODUCT-PRODUCER (P-P)

Ex 20: Organic sesame oil has an anti-bacterial and anti-inflammatory effect.

Gold class: ENTITY-ORIGIN (E-O)

In contrast to Sem10, DDI is composed of extracts from the real-world biomedical domain, where entities are specialists' terms that refer to drug names. Hence, the pre-trained nature of base BERT has a lesser leverage compared to other encoding models. This results in a minimal fluctuation of the results between different encoding systems (as shown in Table 4.11) for DDI. However, the MeasEval data set falls into an in-between category of DDI and Sem10, containing normal English paragraphs, where measured entities (ME) or measured properties (MP) might sometime be scientific terminologies or normal English words, and quantities are always numbers, causing different encoding systems to fluctuate lesser than Sem10 and higher than DDI as pointed out earlier.

The 6TS architecture with IM as input is a good baseline: As can be observed from Table 4.11, the transformer encoder stack performs well on all data sets placing second (except Sem10, as it lacks the pre-trained knowledge contained in BERT), and has either equal or higher precision than the best performing model BERT_{IM}. This shows that the transformer stack can utilize the linguistic features encoded within the *IM* to construct encoding that can be used to perform the RE task and is suggestive of a good architecture to experiment with as a baseline.

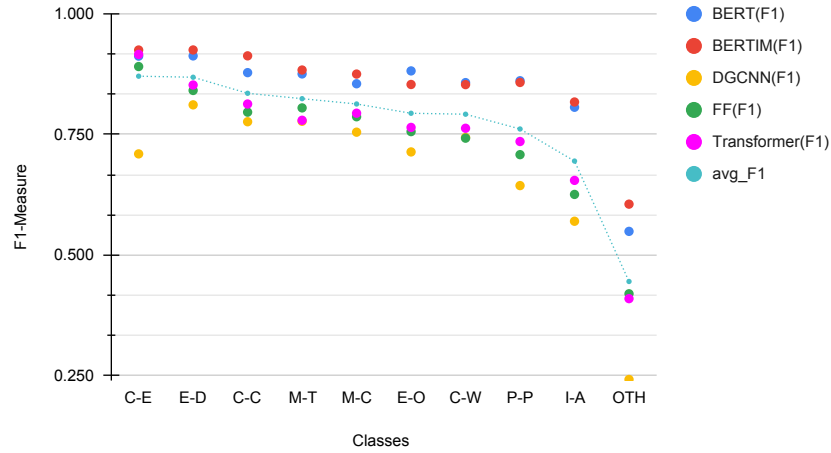


Figure 4.9: Line chart illustrating F1 measure of different encoding schema for each class in Sem10 data set

Error analysis of different encoding systems and Sem10 classes: Figure 4.9 shows the F1 score of each Sem10 class for each encoding schema shown in Table 4.11. As explained above, BERT IM performs comparatively better than BERT on all classes except for the ENTITY-ORIGIN (E-O) class. When analyzing the instances of the (E-O) class for which BERT predicted correctly and BERT IM predicted wrong, out of the 14 instances (see Table 4.5 and Table 4.6) 7 instances were predicted as the OTHER class and some were confused with PRODUCT-PRODUCER (P-P). Ex 21 is an instance BERT IM confused with P-P class. However, compared to other P-P examples provided in the training data (such as example Ex 22 and Ex 23), Ex 21 is not much different, causing confusion.

Ex 21: At one time, most of Europe’s ammonia was produced from the Hydro plant at Vemork, via the electrolysis route.

Gold class: (E-O), BERT IM: (P-P), BERT: (E-O)

Ex 22: The researchers produced a report which has, literally, saved their jobs.

Gold class: (P-P), BERT IM: (P-P), BERT: (P-P)

Ex 23: the Italian manager has drawn up a list of potential targets to reinforce his depleted defensive options.

Gold class: (P-P), BERT IM: (P-P), BERT: (P-P)

Apart from the comparison of BERT and BERT IM, when comparing the BERT-based models with non-BERT models (FF, 6TS, and DGCNN) from Figure 4.9, it is observed that all 3 non-BERT models fall below the average F1 trend line, except for the CAUSE-EFFECT (C-E) class. Additionally, the gap between different classes keeps on widening from left

to right on the x axis. As mentioned earlier, Sem10 is a language-oriented task that relies on word meaning and real-world relationship, hence the gap between BERT-based models and non-BERT models is due to the pre-trained knowledge of BERT. However, for class C-E, all models except for DGCNN have a very high and approximately similar F1 value (BERT = 0.913, BERT IM = 0.925, FF = 0.891 and 6TS = 0.916). When analyzing why this behavior was so, I was able to find that the number of training instances in the training data for this class is high, as shown in Table 4.12 and the training data provided for this particular class have clear patterns that allow them to be easily distinguished from other classes. Similarly to Ex 24 and Ex 25, words / phrases such as *cause*, *caused*, *resulted*, *outputting*, *caused by* and etc. were commonly observed in the training and testing samples provided for the C-E class.

Ex 24: Traffic vibrations on the street outside had caused the movement of the light.

(Gold class: (C-E))

Ex 25: The aircraft was written off in the accident due to the severe impact caused by the KLM aircraft, and the resulting fire.

(Gold class: (C-E))

Table 4.12: Distribution of classes in Sem10 data set

Class	Abbreviation	Training	Testing
OTHER	OTH	1389	449
CAUSE-EFFECT	C-E	1003	328
COMPONENT-WHOLE	C-W	928	306
ENTITY-DESTINATION	E-D	845	292
ENTITY-ORIGIN	E-O	712	257
MEMBER-COLLECTION	M-C	688	231
PRODUCT-PRODUCER	P-P	649	220
MESSAGE-TOPIC	M-T	634	261
CONTENT-CONTAINER	C-C	539	192
INSTRUMENT-AGENCY	I-A	487	152

This behavior of C-E performing equally well for all encoding models (except DGCNN) shows that BERT’s pretrained nature does not offer great advantage in this particular scenario, when the number of training instances is high and when the class has clear patterns that distinguish the class from the others.

Based on Table 4.13, which shows the ranking of different classes in the Sem10 data set for each encoding system, E-D is the class that performs best next to C-E. After analyzing the prediction results of all encoding systems, I observed a similar pattern to the C-E where words/phrases such as *released into*, *into*, *added to the*, *moving into*, *delivered to*, *placed in*

and etc. were observed, as shown in examples Ex 26 and Ex 27 and the number of training instances is also high compared to other classes.

Ex 26: He put a book into the cupboard.

Gold class: (E-D)

Ex 27: All LibriVox recordings have been released into the public domain

Gold class: (E-D)

Table 4.13: Ranking of Sem10 classes based on the F1 score for each encoding system

Model	Rank									
	1	2	3	4	5	6	7	8	9	10
BERT	E-D, C-E		E-O	C-C	M-T	P-P	C-W	M-C	I-A	Oth
BERT_IM	E-D, C-E		C-C	M-T	M-C	P-P	E-O	C-W	I-A	Oth
DGCNN _{mod}	E-D	M-T, C-C		M-C	C-W	E-O	C-E	P-P	I-A	Oth
FF _{row_concat}	C-E	E-D	M-T	C-C	M-C	E-O	C-W	P-P	I-A	Oth
6TS _{no_position}	C-E	E-D	C-C	M-C	M-T	E-O	C-W	P-P	I-A	Oth

Similarly to the C-E and E-D classes consistently ranking in the first two positions in all encoding systems, INSTRUMENT-AGENCY (I-A) and OTHER consistently drop in the last two positions for all systems, as shown in Table 4.13. Among all classes, I-A has the least amount of training data (see Table 4.12), causing all systems to perform poorly. Although the OTHER class, in contrast to I-A, has the most training data samples compared to all classes, the lack of definition of this class causes it to perform consistently poorly in all systems. In other words, all data samples that do not belong to a class fall into the OTHER class, making all models confused as to what to consider as OTHER.

Apart from the classes discussed above (such as C-E, E-D, I-A, and OTHER), the rest of the classes in Sem10 fluctuate between different ranking for different models. However, based on Table 4.5, we can observe that certain classes were confused with other classes, since the definitions of these classes and the provided examples are very similar to each other. Table 4.14 shows a summary of classes confused with each other with an example associated to each class.¹⁵

DGCNN pattern differently: In addition to the ranking of different classes discussed above, another interesting observation in Figure 4.9 is that DGCNN has a strange trend compared to other encoding systems. All encoding systems perform very well in the C-E class, where DGCNN performs very poorly (it falls in the 7th position, while for others it

¹⁵Since the OTHER class is an accumulation of all outlier examples, all classes are confused with the OTHER class; therefore, I omitted it from the summary

Table 4.14: Summary of classes confused with other classes based on confusion matrix of BERT (Table 4.5) with examples; C-E: Cause-Effect, I-A: Instrument-Agency, P-P: Product-Producer, C-C: Content-Container, E-O: Entity-Origin, E-D: Entity-Destination, C-W: Component-Whole, M-C: Member-Collection, M-T: Message-Topic

Class	Confused with	Example
I-A	P-P	The <u>manufacturer</u> assembles the order using <u>parts</u> supplied by his preferred supplier , and ships the order to the retailer .
	C-W	They chose a <u>shop</u> that oven bakes the paint in a down draft paint <u>booth</u>
P-P	I-A	It is used by the <u>grasshoppers</u> to force a <u>burrow</u> in the earth to receive the eggs
	E-O	The <u>license</u> is issued from the new <u>state</u> after verifying the particulars of the applicant license from the original issuing state
C-C	E-D	That measured moisture content of bagged <u>samples</u> inserted in a <u>beaker</u> of soil at various depths
	C-W	A central vacuum is a vacuum <u>motor</u> and filtration system built inside a <u>canister</u>
E-O	C-E	It was windy and cold yesterday and there is still some <u>remnants</u> of snow left from early yesterdays <u>storm</u>
	P-P	Here is a reliable and tasty <u>cookie recipe</u> from my <u>grandmother</u>
E-D	C-C	Douglas obeyed the king , and the <u>heart</u> was enclosed in a silver <u>casket</u>
C-W	I-A	The <u>film</u> uses <u>flashbacks</u> as a device to tell the story , which was based on a 1947 novel by David Goodis
	C-C	This SNES controller has a <u>flash</u> drive loaded with <u>ROMs</u> inside
	M-T	This <u>page</u> carries a list of some of these <u>conventions</u>
M-C	M-T	Carolyn Cooke’s <u>stories</u> have been featured in the <u>book</u>
M-T	M-C	A <u>calendar</u> of <u>saints</u> forms a way of organising a liturgical year on the finely-granulated level of days by assigning each day to association with a saint
	C-W	The album contains not only songs but also spoken <u>parts</u> narrating parts of the <u>story</u>

falls in the 1st position). Similarly, based on Table 4.13, the M-T class falls to the 2nd position in DGCNN, when it is in the 4th or 5th position for all other models. Also, when considering the difference in the F1 score between the first and last classes, DGCNN shows a 70% drop, while the BERT and non-BERT systems show a difference between 40-50%. Even though I do not have an explanation as to why this behavior, this observation raises some questions as to whether DGCNN is an appropriate architecture to use for NLP tasks especially relation extraction task? Or does the naive way of my implementation not reveal the potential of a graph-based approach on relation extraction task?

Leaving the Sem10 data set aside, taking into account the recall values observed for the DGCNN system in Table 4.11, which is almost the second highest compared to other encoding schema, I lean more toward the discussion of improving the DGCNN model than completely removing it from experimentation. Even though the DGCNN system has poor Precision, the high Recall shows that the system is capable of addressing the task, but with more experimentation to improve the precision.

BERT-based models versus FF on DDI: Moving away from Sem10, when observing the F1 scores of the other two data sets in Table 4.11, regardless of the complexity of the models *FFrow_concat* (approximately 6.8M parameters) and BERT (110M parameters), *FFrow_concat* shows a marginal increase in performance compared to BERT for DDI, and much closer results for MeasEval. However, BERT_{IM} shows improvement over BERT, concluding that the BERT model benefits from *IM*, and *IM* is not confounding to BERT. To understand why *FFrow_concat* performs better compared to BERT for DDI, the experiments shown in Table 4.15 were carried out to evaluate the following hypothesis.

- Hypothesis 1: The *IM* contains PubMed ELMo embeddings, which are pre-trained on PubMed abstracts (biomedical texts). Since DDI and MeasEval contain abstracts or sentences from scientific articles, the use of PubMed ELMo embedding over the base BERT gives an additional advantage to FF to leverage it and perform comparatively equivalent to BERT.
- Hypothesis 2: The *IM* consists of 3 other features (POS tags, Dependency path, Distance feature) apart from the token embedding. Adding these features contributes to the performance of FF being equivalent to that of BERT.

Based on the hypothesis, to evaluate the influence of token embeddings, experiments were carried out in FF replacing the PubMed ELMo embedding (FF_{pubmed}) with the original ELMo embedding (FF_{org}). Furthermore, to evaluate the influence of other features except

token embedding, IM with all features (FF(All)) and IM with only token embedding (FF(Token)) were carried out.

Table 4.15: Precision(P), Recall(R) and F1-measure(F1) of all 3 data sets to evaluate the impact of PubMed ELMo (FF_{pubmed}) versus Original ELMo (FF_{org}), and token embedding (FF (Token)) versus all features (FF (All)) in a FF network

Experiment	Sem10			DDI			MeasEval - MP			MeasEval-ME		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
FF _{org} (All)	.77	.77	.77	.81	.84	.83	.68	.67	.66	.67	.67	.67
FF _{pubmed} (All)	.74	.74	.74	.84	.86	.85	.70	.67	.68	.69	.69	.69
FF _{org} (Token)	.77	.77	.77	.83	.82	.82	.67	.64	.65	.66	.65	.65
FF _{pubmed} (Token)	.73	.73	.73	.80	.85	.82	.67	.67	.67	.66	.67	.66

Taking into account the results of FF_{org} (All) vs FF_{pubmed} (All) or FF_{org} (Token) vs FF_{pubmed} (Token), except for the Sem10 data set, FF_{pubmed} performs comparatively better than FF_{org} for both the DDI and MeasEval data sets. It proves that Hypothesis one is right and that PubMed ELMo embedding does influence the results for data sets based on scientific data. Furthermore, the contrast behavior of Sem10 shows that the embedding of PubMed ELMo hinders performance when used with ordinary English data sets.

Regarding the second hypothesis, the comparison of FF_{org} (All) versus FF_{org} (Token) or FF_{pubmed} (All) versus FF_{pubmed} (Token) shows that FF(All) performs better than FF(Token) for DDI and MeasEval, but does not influence Sem10. This may be because Sem10 is a language-oriented task that relies on the meanings of actual words and relationships in the real world (as discussed earlier), and the linguistic features encoded within IM are not discriminatory in distinguishing different classes. Therefore, external linguistic features such as POS and dependency relationship do not influence much when token embedding such as ELMo contain pre-trained knowledge, which has greater influence than of hand-crafted features. However, for DDI and MeasEval, the inclusion of linguistic features in IM is beneficial because the data set does not rely heavily on the meaning of words, rather the context and connectivity. Therefore, Hypothesis 2 is true for DDI and MeasEval, while it is false for Sem10.

4.9 Re-evaluating the IM matrix

The following sections discuss additional experiments conducted on Sem10 and DDI to re-evaluate dependency-based features included in the IM¹⁶.

¹⁶Due to limited time and difficulty in analyzing, MeasEval data set was omitted from experiments

4.9.1 Dependency-based IM (IM_{dep}) versus Sentence-based IM (IM_{sent})

The experiments discussed above are based on an IM matrix constructed on the basis of the connectivity of entities in the dependency parse tree. Although experiments were carried out on different encoding systems, all use the same IM in one way or another. Constructing IM restricted to tokens between entities based on a dependency parse tree leads to a large preprocessing step prior to training a classification model. To evaluate whether this painstaking preprocessing is actually beneficial rather than encoding all tokens in the sentence in a matrix IM , several experiments indicated in Table ?? were performed. Instead of running on all encoding systems discussed above, I chose DGCNN (since it is the basic architecture that motivated the IM data structure) and BERT- IM (since it is the encoding system that performs well compared to all my other models). The features within the IM for each token are exactly the same as those used in all the above experiments (that is, the feature vector of each token is a concatenation of one hot POS vector, the PubMed ELMo embedding, the Node2Vec dependency path embedding from entity 1, and the distance feature from the target entities (see Section 3.3)).

Table 4.16: Precision (P), Recall (R), and F1 measure (F1) of Sem10 and DDI data set to evaluate sentence-based IM (IM_{sent}) versus dependency-based IM (IM_{dep}) on the DGCNN and BERT- IM encoding systems

Experiment	Sem10			DDI		
	P	R	F1	P	R	F1
DGCNN _{dep}	.68	.73	.69	.81	.88	.84
DGCNN _{sent}	.61	.64	.60	.74	.84	.77
BERT- IM_{dep}	.85	.86	.85	.85	.90	.87
BERT- IM_{sent}	.85	.87	.85	.74	.83	.77

Based on Table ??, IM_{dep} performs better on both DGCNN and BERT- IM with the exception of BERT- IM_{sent} for Sem10, which has a marginal increase in precision. This may be because the Sem10 training data are simple sentences with only a pair of entities annotated. In IM_{sent} , IM has more context annotated than IM_{dep} , providing an added advantage for BERT. However, considering the memory consumption required to process IM_{sent} compared to IM_{dep} , this increase in precision is negligible. Unlike Sem10, DDI shows an approximately 10% difference in F1 for DDI, indicating that IM_{sent} not only consumes more memory to process, but also contains unnecessary information that hinders the performance of BERT. Therefore, in summary, IM_{dep} shows more potential than IM_{sent} for the DDI task; on the contrary, for a language-oriented task like Sem10 and especially the way the data are provided, IM_{sent} shows a very minimal increase. However, considering the memory consumption of both matrices, IM_{dep} seems more promising.

4.9.2 Dependency path versus Dependency governor

Among the many different ways to represent the dependency relationship, I chose to represent the dependency path by concatenating the Node2Vec embedding of the dependency relationship when traversing from entity 1 to the target token (see Section 3.3.4) in IM . To evaluate whether it is an effective representation or not, I replaced the path information with the Node2Vec embedding of the immediate governor relationship to assess whether the naive representation of dependency is sufficient or not. Similarly to the discussion above, I experimented with DGCNN and BERT_IM and chose IM_{dep} as it showed promising results.

Table 4.17: Precision (P), Recall (R) and F1 measure (F1) of Sem10 and DDI data sets to evaluate the dependency path encoding (dep_path) against dependency governor relationship (dep_gov)

Experiment	Sem10			DDI		
	P	R	F1	P	R	F1
DGCNN(dep_gov)	.68	.74	.69	.77	.86	.80
DGCNN(dep_path)	.68	.73	.69	.81	.88	.84
BERT_IM(dep_gov)	.85	.86	.85	.84	.88	.86
BERT_IM(dep_path)	.85	.86	.85	.85	.90	.87

Table ?? shows the results of the experiments conducted on DGCNN and BERT_IM with varying dependency relationship encoding. According to the table, DDI seems to benefit more from the path information than the governor alone in both models. However, Sem10 does not show any change in BERT_IM, since BERT_IM does not seem to rely on the dependency information (based on Table 4.15), validating the discussion more with Table 4.15. However, there is a minimal increase in precision in $DGCNN_{dep}(dep_gov)$ than $DGCNN_{dep}(dep_path)$. In conclusion, the dep_path representation of dependency relationship between entities contributes to the classification task, especially for the DDI data set.

Chapter 5

Conclusion and Future Work

Current NLP research is driven by deep learning models, leading to a shift of focus from the use of linguistic features as the primary source of input. This thesis focuses on addressing the Relation Extraction (RE) task represented with basic linguistic features and provided as input to different encoding schema, including transformer-based deep models such as BERT, to evaluate the impact of linguistic features in the RE task. Although specific linguistic features were chosen to be represented in certain ways, the thesis does not emphasize the specificities of the features but rather the impact of enabling the linguistic features to be represented in different encoding schemas. Experiments were conducted with three different data sets, Sem10, DDI, and MeasEval, which vary in complexity and in nature of the text. Sem10 is a SemEval 2010 data set consisting of ordinary English sentences with nine types of relations and an additional OTHER type. The nine types of relations are: CAUSE-EFFECT, INSTRUMENT-AGENCY, PRODUCT-PRODUCER, CONTENT-CONTAINER, ENTITY-ORIGIN, ENTITY-DESTINATION, COMPONENT-WHOLE, MEMBER-COLLECTION, and MESSAGE-TOPIC. DDI is a SemEval 2013 data set that contains texts from the biomedical domain in which the interaction between drugs should be predicted as TRUE or FALSE. MeasEval is a SemEval 2021 task consisting of scientific excerpts, from which quantities/measurements should be identified along with the entities that are being measured (MEASUREDENTITY) and properties of the measured entity if there exist any (MEASUREDPROPERTY). Unlike Sem10 and DDI, MeasEval does not explicitly annotate its entities. Therefore, the original task is designed to separate the task into several subtasks to identify entities and classify relationships based on identified entities. However, instead of the original approach to the task, the MeasEval task was performed in a way that performed both entity detection and relation extraction simultaneously in this thesis. Since the main focus of this thesis is relation extraction, all subtasks associated with each data set (excluding

relation extraction) were omitted from the discussion.

The main objective of an RE task is to predict the relationship between two entities. Since dependency parse trees are capable of providing approximations of semantic relationships between predicates and their arguments, my main interest was to generalize over dependency parse trees to obtain latent path representations to distinguish different semantic connections between the target entities.

Since dependency parse trees are in a form of graph representation, I attempted to use a Graph Neural Network (GNN) in the form of a Deep Graph Convolution Neural Network (DGCNN) to extract latent features from the dependency parse tree to distinguish between different relationships. Even though it seemed logical to utilize a GNN based network on the dependency parse tree, the results obtained on all three datasets do not show promising results and perform somewhat poorly compared to other experiments discussed in this thesis. This observation raises multiple questions, which I find interesting to explore in future work. First, is the DGCNN architecture chosen for this task a wise choice or not? Even though there are many GNN based architectures, I chose DGCNN because it was showing some promising results on the link prediction task (on graph-based data sets), which I find to be similar to the RE task. Second, is the implementation of the DGCNN aligned with the requirement of an RE task? Although I experimented by changing the number and size of graph convolutions in the DGCNN architecture, I did not explore the other parameters, which could have an impact on the results. I raise this question since the change in convolution showed an approximately 4% increase from the original parameter setting in DDI and Measeval, while showing an approximately 13% increase in the Sem10 data set. Furthermore, compared to other encoding schema, the DGCNN system had the second highest recall value on all data sets. This indicates that the system is capable of addressing the RE task, which requires more experimentation to improve precision. Based on a review of the literature related to the GNN model, [Y. Zhang et al. \(2018\)](#) and [D. Li and Ji \(2019\)](#) combine the representation of the graph with the representation of the target entities before feeding it to a classifier for prediction. This I find interesting to explore as a future work, since my classifier only uses the graph representation, whereas a classifier could benefit from explicitly provided target entity representation.

The DGCNN architecture accepts input in the form of an Adjacency matrix (A) and an Information matrix (IM). The matrix A represents the connectivity of the tokens (between the two target entities) based on the dependency parse tree. Although the matrix A contains token connectivity, it does not resemble the type of connection. Since dependency parse trees have typed relationships, future experiments can be conducted to reflect the type in the matrix A by providing different weights for different types of edges. For each corresponding

row in A , IM will contain a row representing each token in the dependency parse tree. To maintain a general approach across the different data sets, this representation is constructed by encoding basic linguistic features that are commonly observed on all data sets. However, these features are not the only features that could be used, and this can be extended with additional linguistic features that are specific to the task to improve the performance of the different encoding schema discussed in this thesis.

To encode the linguistic features obtained from the dependency parse tree, I experimented with a few machine learning architectures that are widely used in the field of NLP research by providing IM as input, primary, or complementary. These neural architectures include classical baselines such as Random Forest (RF), Support Vector Machines (SVM), and Feed-Forward Neural Network (FF), and trending baseline models such as BERT and Transformers. All models except BERT accept IM as the primary input, while for BERT (BERT_{IM}), IM is used as the complementary input to its classifier by a simple concatenation with the [CLS] token obtained from the last layer of BERT.

On the basis of the experiment results on all data sets, the difference in performance between BERT-based models and non-BERT-based models indicates a clear distinction between the different data sets. Compared to FF, Sem10 shows a 10% increase with BERT, while DDI shows a 1% drop in performance with BERT. This clearly indicates that while BERT’s pre-training yields significant benefits for Sem10 than the non-language-oriented DDI task. To be more precise, BERT in DDI shows the lowest performance, indicating that the off-the-shelf base BERT model does not have a clear advantage, regardless of its complexity (110M parameters) in all data sets. Although it was not tested in this thesis, domain-specific language models such as BioBERT (Lee et al., 2019) or SciBERT (Beltagy, Lo, & Cohan, 2019) might have an added advantage on the DDI and MeasEval data sets, in comparison to base BERT. Even though this is an hypothesis and should be tested, this indicates that a significant amount of overhead has to be applied to pre-train these language models to cater different genres of texts, where simple concatenation of linguistic features may perform equally well on data sets similar to DDI.

One of the main disadvantages I encountered while using pre-trained models like BERT is that external features cannot be directly incorporated without taking the BERT implementation apart. In this thesis, I attempt to evaluate a BERT-like architecture with external features with two strategies: One, mimic the BERT architecture with a transformer encoder stack by providing IM as the primary input. Two, directly concatenating IM to [CLS] of BERT prior to classification (BERT_{IM}). The transformer stack performs second, with either higher or equal precision to BERT_{IM} on all datasets except Sem10 (as the 6TS lacks the pre-trained knowledge which is more effective on this task). Indicating

that the transformer architecture can take advantage of the linguistic features to perform the RE task. However, considering the F1 score, BERT_{IM} performs consistently better on all three data sets compared to all the experiments discussed in this thesis. This shows that incorporating linguistic features improves the pre-trained model and is not confounding. Although the increase in BERT_{IM} performance is marginal compared to BERT, it can be attributed to very basic linguistic features in *IM*. This should be explored with more task-specific features, such as providing trigger words for Sem10 data points to distinguish between different classes or incorporating information from external sources for drugs in DDI. However, careful attention should be paid to how these linguistic features are chosen and how they are represented, as experiments with *IM* constructed on the whole sentence degrade the performance of the models, while dependency-based *IM* performs well. Also, providing dependency path information contributes more to DDI than just providing dependency relationship to the immediate governor. In addition, the way the external feature was incorporated with BERT was by a simple concatenation. This can be explored with an implementation of having a layer of self-attention on top of [CLS] and *IM* to obtain a better representation before classification.

In conclusion, based on the experiments on the selected data set, linguistic features based on the dependency parse tree show competitive performance on simpler models such as FF compared to complex models such as BERT, and combining linguistic features with simple concatenation with BERT improves the performance of the BERT architecture.

References

- Atwood, J., & Towsley, D. (2016). Diffusion-Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 29 (NIPS'16)*.
- Bahdanau, D., Cho, Hyun, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, (ICLR'15)* .
- Baldini Soares, L., FitzGerald, N., Ling, J., & Kwiatkowski, T. (2019). Matching the Blanks: Distributional Similarity for Relation Learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL'19)*.
- Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton University Press.
- Beltagy, I., Lo, K., & Cohan, A. (2019). SciBERT: Pretrained Language Model for Scientific Text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP'19)*.
- Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3.
- Bethard, S., Savova, G., Chen, W.-T., Derczynski, L., Pustejovsky, J., & Verhagen, M. (2016). SemEval-2016 Task 12: Clinical TempEval. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*.
- Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python — Analyzing Text with the Natural Language Toolkit*. O'Reilly Media.
- Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.
- Bobić, T., Fluck, J., & Hofmann-Apitius, M. (2013). SCAI: Extracting drug-drug interactions using a rich feature vector. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval-2013)*.
- Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., & Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21.

- Bruna, J., Zaremba, W., Szlam, A., & Lecun, Y. (2014). Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR'14)*.
- Bunescu, R., & Mooney, R. (2005). A Shortest Path Dependency Kernel for Relation Extraction. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (EMNLP'05)*.
- Cai, R., Zhang, X., & Wang, H. (2016). Bidirectional Recurrent Convolutional Neural Network for Relation Classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL'16)*.
- Cheng, J., Dong, L., & Lapata, M. (2016). Long Short-Term Memory-Networks for Machine Reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP'16)*.
- Chowdhury, M. F. M., & Lavelli, A. (2013). FBK-irst : A Multi-Phase Kernel Based Approach for Drug-Drug Interaction Detection and Classification that Exploits Linguistic Information. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval-2013)*.
- Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, 20.
- Culotta, A., McCallum, A., & Betz, J. (2006). Integrating Probabilistic Extraction Models and Data Mining to Discover Relations and Patterns in Text. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL:HLT'06)*.
- Cunningham, H., Maynard, D., Bontcheva, K., & Tablan, V. (2002). GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*.
- Cunningham, H., Tablan, V., Roberts, A., & Bontcheva, K. (2013). Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics. *PLoS Computational Biology*, 9(2).
- Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., & Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34.
- Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems 29 (NIPS'16)*.

- de Marneffe, M.-C., MacCartney, B., & Manning, C. D. (2006). Generating Typed Dependency Parses from Phrase Structure Parses. In *The International Conference on Language Resources and Evaluation (LREC'06)*.
- de Marneffe, M.-C., Manning, C. D., Nivre, J., & Zeman, D. (2021). Universal Dependencies. *Computational Linguistics*, 47.
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL:HLT'19)*.
- Dobson, P. D., & Doig, A. J. (2003). Distinguishing Enzyme Structures from Non-enzymes Without Alignments. *Journal of Molecular Biology*, 330.
- Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., & Adams, R. P. (2015). Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in Neural Information Processing Systems 28 (NIPS'15)*.
- Ebrahimi, J., & Dou, D. (2015). Chain Based RNN for Relation Classification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'15)*.
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14.
- Gao, T., Han, X., Zhu, H., Liu, Z., Li, P., Sun, M., & Zhou, J. (2019). FewRel 2.0: Towards More Challenging Few-Shot Relation Classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP'19)*.
- Gast, V., Bierkandt, L., Druskat, S., & Rzymiski, C. (2016). Enriching TimeBank: Towards a more precise annotation of temporal relations in a text. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*.
- Grishman, R. (1997). Information extraction: Techniques and challenges. In *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology*.
- Grover, A., & Leskovec, J. (2016). node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*.
- Han, X., Zhu, H., Yu, P., Wang, Z., Yao, Y., Liu, Z., & Sun, M. (2018). FewRel:A Large-Scale Supervised Few-Shot Relation Classification Dataset with State-of-the-Art Evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP'18)*.
- Harper, C., Cox, J., Kohler, C., Scerri, A., Daniel Jr., R., & Groth, P. (2021). SemEval 2021

- Task 8: MeasEval – Extracting Counts and Measurements and their Related Contexts. In *Proceedings of the 5th Workshop on Semantic Evaluation (SemEval-2021)*.
- Hendrickx, I., Kim, S. N., Kozareva, Z., Nakov, P., Ó Séaghdha, D., Padó, S., . . . Szpakowicz, S. (2010). SemEval-2010 Task 8: Multi-Way Classification of Semantic Relations between Pairs of Nominals. In *Proceedings of the 5th International Workshop on Semantic Evaluation (SemEval-2010)*.
- Hinton, G. E., McClelland, J. L., & Rumelhart, D. E. (1986). Distributed Representations. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*.
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9.
- Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL'18)*.
- Jurafsky, D., & Martin, J. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson Prentice Hall.
- Kambhatla, N. (2004). Combining Lexical, Syntactic, and Semantic Features with Maximum Entropy Models for Extracting Relations. In *Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions (ACL'04)*.
- Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*.
- Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, (ICLR'15)*.
- Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR'17)*.
- Klein, D., & Manning, C. D. (2003). Accurate Unlexicalized Parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics (ACL'03)*.
- Krebs, A., Lenci, A., & Paperno, D. (2018). SemEval-2018 Task 10: Capturing Discriminative Attributes. In *Proceedings of The 12th International Workshop on Semantic Evaluation (SemEval-2018)*.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25 (NIPS'12)*.

- Lai, S., Leung, K. S., & Leung, Y. (2018). SUNNYNLP at SemEval-2018 Task 10: A Support-Vector-Machine-Based Method for Detecting Semantic Difference using Taxonomy and Word Embedding Features. In *Proceedings of The 12th International Workshop on Semantic Evaluation (SemEval-2018)*.
- LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object Recognition with Gradient-Based Learning. In *Shape, Contour and Grouping in Computer Vision*.
- Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2019). BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4).
- Levy, O., Seo, M., Choi, E., & Zettlemoyer, L. (2017). Zero-Shot Relation Extraction via Reading Comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL'17)*.
- Li, D., & Ji, H. (2019). Syntax-aware Multi-task Graph Convolutional Networks for Biomedical Relation Extraction. In *Proceedings of the Tenth International Workshop on Health Text Mining and Information Analysis (LOUHI'19)*.
- Li, J., Sun, Y., Johnson, R. J., Sciaky, D., Wei, C.-H., Leaman, R., ... Lu, Z. (2016). BioCreative V CDR task corpus: a resource for chemical disease relation extraction. *Database*, 2016.
- Lin, Y., Shen, S., Liu, Z., Luan, H., & Sun, M. (2016). Neural Relation Extraction with Selective Attention over Instances. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL'16)*.
- Liu, W., Fu, X., Zhang, Y., & Xiao, W. (2021). Lexicon Enhanced Chinese Sequence Labeling Using BERT Adapter. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP'21)*.
- Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL'14)*.
- Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19.
- McDonald, R., Pereira, F., Kulick, S., Winters, S., Jin, Y., & White, P. (2005). Simple Algorithms for Complex Relation Extraction with Applications to Biomedical IE. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word

- Representations in Vector Space. In *1st International Conference on Learning Representations, (ICLR'13)*.
- Miwa, M., & Bansal, M. (2016). End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL'16)*.
- Nguyen, T. H., & Grishman, R. (2015). Relation Extraction: Perspective from Convolutional Neural Networks. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*.
- Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (IEEE'14)*.
- Pal, S., & Mitra, S. (1992). Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on Neural Networks*, 3.
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL:HLT'18)*.
- Qian, L., Zhou, G., Kong, F., Zhu, Q., & Qian, P. (2008). Exploiting Constituent Dependencies for Tree Kernel-Based Semantic Relation Extraction. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING'08)*.
- Radford, A., & Narasimhan, K. (2018). Improving Language Understanding by Generative Pre-Training. *Preprint*.
- Riedel, S., Yao, L., & McCallum, A. (2010). Modeling Relations and Their Mentions without Labeled Text. In *The European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases (ECML PKDD'10)*.
- Rink, B., & Harabagiu, S. (2010). UTD: Classifying Semantic Relations by Combining Lexical and Semantic Resources. In *Proceedings of the 5th International Workshop on Semantic Evaluation (SemEval-2010)*.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323.
- Segura-Bedmar, I., Martínez, P., & Herrero-Zazo, M. (2013). SemEval-2013 Task 9 : Extraction of Drug-Drug Interactions from Biomedical Texts (DDIExtraction 2013). In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval-2013)*.
- Segura-Bedmar, I., Martinez, P., & Sanchez-Cisneros, D. (2011). The 1st DDIExtraction-2011 Challenge Task: Extraction of Drug-Drug Interactions from Biomedical Texts.

- In *Proceedings of the 1st Challenge Task on Drug-Drug Interaction Extraction 2011*.
- Shen, Y., & Huang, X. (2016). Attention-Based Convolutional Neural Network for Semantic Relation Extraction. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers (COLING'16)*.
- Socher, R., Huval, B., Manning, C. D., & Ng, A. Y. (2012). Semantic Compositionality through Recursive Matrix-Vector Spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP'12)*.
- Sorokin, D., & Gurevych, I. (2017). Context-Aware Representations for Knowledge Base Relation Extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP'17)*.
- Speer, R., & Lowry-Duda, J. (2018). Luminoso at SemEval-2018 Task 10: Distinguishing Attributes Using Text Corpora and Relational Knowledge. In *Proceedings of The 12th International Workshop on Semantic Evaluation (SemEval-2018)*.
- Suchanek, F. M., Ifrim, G., & Weikum, G. (2006). Combining Linguistic and Statistical Analysis to Extract Relations from Web Documents. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*.
- Thillaisundaram, A., & Togia, T. (2019). Biomedical relation extraction with pre-trained language representations and minimal task-specific architecture. In *Proceedings of The 5th Workshop on BioNLP Open Shared Tasks (BioNLP'19)*.
- Thomas, P., Neves, M., Rocktäschel, T., & Leser, U. (2013). WBI-DDI: Drug-Drug Interaction Extraction using Majority Voting. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval-2013)*.
- Urooj, A., Mazaheri, A., Da vitoria lobo, N., & Shah, M. (2020). MMFT-BERT: Multi-modal Fusion Transformer with BERT Encodings for Visual Question Answering. In *Findings of the Association for Computational Linguistics (EMNLP'20)*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems 30 (NIPS'17)*.
- Wang, L., Cao, Z., de Melo, G., & Liu, Z. (2016). Relation Classification via Multi-Level Attention CNNs. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL'16)*.
- Wishart, D. S., Knox, C., Guo, A. C., Shrivastava, S., Hassanali, M., Stothard, P., ... Woolsey, J. (2006). DrugBank: a comprehensive resource for in silico drug discovery and exploration. *Nucleic Acids Research*, 34.

- Wu, S., & He, Y. (2019). Enriching Pre-trained Language Model with Entity Information for Relation Classification. *arXiv preprint arXiv:1905.08284*.
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... others (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.
- Yanardag, P., & Vishwanathan, S. (2015). Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*.
- Yang, A., Wang, Q., Liu, J., Liu, K., Lyu, Y., Wu, H., ... Li, S. (2019). Enhancing Pre-Trained Language Representations with Rich Knowledge for Machine Reading Comprehension. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL'19)*.
- Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems 32 (NeurIPS'19)*.
- Yao, Y., Ye, D., Li, P., Han, X., Lin, Y., Liu, Z., ... Sun, M. (2019). DocRED: A Large-Scale Document-Level Relation Extraction Dataset. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL'19)*.
- Zeng, D., Liu, K., Lai, S., Zhou, G., & Zhao, J. (2014). Relation Classification via Convolutional Deep Neural Network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers (COLING'14)*.
- Zhang, M., & Chen, Y. (2018). Link Prediction Based on Graph Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS'18)*.
- Zhang, M., Cui, Z., Neumann, M., & Chen, Y. (2018). An End-to-End Deep Learning Architecture for Graph Classification. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18)*.
- Zhang, Y., Qi, P., & Manning, C. D. (2018). Graph Convolution over Pruned Dependency Trees Improves Relation Extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP'18)*.
- Zhang, Y., Zhong, V., Chen, D., Angeli, G., & Manning, C. D. (2017). Position-aware Attention and Supervised Data Improve Slot Filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP'17)*.

Appendix A

Ablation study on linguistic feature representation

This chapter provides a sample of preliminary ablation experiments conducted on different representations of linguistic features. These experiments were carried out on the MeasEval data set on the DGCNN system. In total, 1560 experiments were conducted; however, only a few relevant experiments are highlighted in Table [A.1](#). Although the representation techniques were chosen based on the results of this table, the evaluation script used to produce these results is different from the evaluation scheme used in the thesis content.

Table A.1: Partial results of an ablation study conducted on different representation of linguistic features on MeasEval data set on DGCNN system

Hop_size	POS tag	Dependency	Token	Precision	Recall	F1-measure
8	One-hot	none	none	0.32	0.38	0.35
6	One-hot	none	none	0.33	0.37	0.35
9	One-hot	none	none	0.31	0.37	0.34
7	One-hot	none	none	0.28	0.39	0.33
5	One-hot	none	none	0.28	0.38	0.32
10	Label_Encoded	none	none	0.30	0.33	0.32
6	Label_Encoded	none	none	0.29	0.33	0.31
8	Label_Encoded	none	none	0.30	0.32	0.31
5	Label_Encoded	none	none	0.29	0.33	0.31
7	Label_Encoded	none	none	0.30	0.32	0.31
10	One-hot	none	none	0.27	0.35	0.31
9	Label_Encoded	none	none	0.28	0.33	0.31
8	One-hot	Label_Encoded	Elmo	0.46	0.39	0.42
8	One-hot	Node2Vec_Concat	Elmo	0.42	0.42	0.42
8	none	Label_Encoded	Elmo	0.47	0.38	0.42
8	Label_Encoded	none	Elmo	0.43	0.40	0.41
8	none	none	Elmo	0.44	0.39	0.41
8	none	Multi-hot	Elmo	0.43	0.40	0.41
8	Label_Encoded	Multi-hot	Elmo	0.44	0.39	0.41
8	Label_Encoded	Label_Encoded	Elmo	0.46	0.37	0.41
8	Label_Encoded	Node2Vec_Avg	Elmo	0.43	0.39	0.41
8	Label_Encoded	one_hot	Elmo	0.43	0.39	0.41
8	One-hot	Multi-hot	Elmo	0.43	0.39	0.41
8	none	one_hot	Elmo	0.43	0.39	0.41
8	One-hot	one_hot	Elmo	0.43	0.38	0.40
8	One-hot	none	Elmo	0.42	0.39	0.40
8	none	Node2Vec_Avg	Elmo	0.42	0.39	0.40
8	none	Node2Vec_Concat	Elmo	0.43	0.38	0.40
8	Label_Encoded	Node2Vec_Avg	Elmo	0.42	0.39	0.40
8	One-hot	Node2Vec_Concat	Elmo	0.41	0.37	0.39
8	One-hot	Node2Vec_Concat	none	0.33	0.37	0.35
8	One-hot	Multi-hot	none	0.33	0.37	0.35
8	none	Node2Vec_Concat	none	0.32	0.37	0.34
8	none	Multi-hot	none	0.34	0.35	0.34
8	One-hot	Node2Vec_Avg	none	0.33	0.35	0.34
8	none	Node2Vec_Avg	none	0.33	0.34	0.33
8	One-hot	one_hot	none	0.29	0.37	0.33
8	Label_Encoded	Node2Vec_Avg	none	0.33	0.32	0.32
8	Label_Encoded	Node2Vec_Concat	none	0.32	0.33	0.32
8	Label_Encoded	Multi-hot	none	0.33	0.32	0.32
8	none	One-hot	none	0.30	0.32	0.31
8	Label_Encoded	none	none	0.30	0.32	0.31
8	Label_Encoded	Label_Encoded	none	0.29	0.33	0.31
8	none	Label_Encoded	none	0.31	0.31	0.31
8	One-hot	Label_Encoded	none	0.28	0.34	0.31
8	Label_Encoded	One-hot	none	0.27	0.33	0.30

Appendix B

Official evaluation scheme of the MeasEval task

The official evaluation script¹ expects the system prediction results to be in tab-separated value (.tsv) file format for each document in the test data. Figure B.1 shows a sample of the output structure. Using these files, the evaluation script performs a first pass to match the submission Annotation Set IDs to the corresponding gold set annotation IDs and performs a second pass to evaluate the detection of entities, relationships, and additional details associated with each annotation type against the gold annotation.

The output of the evaluation script is very extensive, providing detailed information on true positives, false positives, and false negatives along with the precision, recall, and F measures for each annotation type and relationship type defined in the task. In addition, the Exact Match (EM) score and the SQuAD-style F1 (overlap) score of the overall system are also outputted based on which the system’s overall performance is evaluated.

The exact match is a binary value of 0 or 1 depending on whether the prediction span exactly matches the gold span, while the F1 overlap is a token level overlap ratio of the system output to the gold span. For components that do not include a span, the exact match and F1 scores are the same. Relations such as HasQuantity, HasProperty, and HasQualifier are also scored with a binary match score if the relation types match and both endpoints match either exactly or with some overlap.

Any span, unit, modifier, or relationship found in the gold data but not the submission, or vice versa, are included as a ”penalty row” with a score to sufficiently penalize both false positives and false negatives when averaging scores. Due to the complex nature of the task and the complex evaluation scheme with penalties, it was decided to modify the evaluation

¹<https://github.com/harperco/MeasEval/tree/main/eval>

docId	annotSet	annotType	startOffset	endOffset	annotId	text	other
S0012821X12004384-952	1	Quantity	249	253	T1-1	<30%	{"mods": ["!sRange"], "unit": "%"}
S0012821X12004384-952	1	MeasuredProperty	254	271	T2-1	wood/plant tissue	{"HasQuantity": "T1-1"}
S0012821X12004384-952	1	MeasuredEntity	236	243	T3-1	samples	{"HasProperty": "T2-1"}
S0012821X12004384-952	2	Quantity	380	384	T1-2	>30%	{"mods": ["!sRange"], "unit": "%"}
S0012821X12004384-952	2	MeasuredProperty	385	402	T3-2	wood/plant tissue	{"HasQuantity": "T1-2"}
S0012821X12004384-952	2	MeasuredEntity	367	374	T2-2	samples	{"HasProperty": "T3-2"}
S0012821X12004384-952	3	Quantity	658	677	T1-3	2614.7 and 2619.6 m	{"mods": ["!sList"], "unit": "m"}
S0012821X12004384-952	3	MeasuredEntity	641	654	T2-3	Values shaded	{"HasQuantity": "T1-3"}

Figure B.1: Format of a .tsv output file expected by the MeasEval evaluation script. *docId*: the document id of the evaluated text; *annotSet*: the annotation set an annotation type belongs to; *annotType*: the annotation type of each detected entity; *startOffset*, *endOffset*: the starting and ending offset respectively of the spans of the entities identified for each annotation type; *text*: resembles the text withing the span of the offsets identified; *other*: includes additional details such as relationship type, if the annotation type is any other than quantity, and for quantity this includes the units and modifier if identified any.

scheme in a more simplistic way to address in this thesis.