Extracting Relational Facts from Plain Text and Building Knowledge Graph Out of Them

by

Mohammad Sahand Parniani

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Graph-based Knowledge Bases (KBs) are composed of relational facts that can be perceived as two entities, called *head* and *tail*, linked through a relation. Processes of constructing KBs, i.e., populating them with such facts, as well as revising and updating them are of special importance. Such tasks require automatic methods and procedures, especially in the case when the main sources of facts are textual documents.

This research aims at applying Machine Learning and Computational Intelligence methods for the analysis of textual data and recommending a methodology for extracting structured information from unstructured text. The goal is to design and propose a method to extract triples from sentences in the form of <*head, relation, tail*>. These extracted triples from the text can be used to build a graph-based KBs or update the existing ones.

For the first part of this research, a task of Relation Extraction (RE), i.e., predicting a relation that links two entities mentioned in a sentence, is investigated. Using RE processes, new relational facts from unstructured texts should be extracted. In this part, we develop a new method for RE which is based on cleaning the input sequence that is fed to the model. This is obtained by removing noisy tokens from the sentence using dependency tree. This helps the model to focus more on the tokens that contribute more to identifying the relation. We also utilize entity type information and inject that to the model to get a better performance. Our method is tested on the widely used NYT dataset and compared to other state-of-the-art methods in RE. Experimental results prove the effectiveness of the developed procedure

compared to other methods.

For the second part of this research, we focus on a triple extraction task. The main difference between triple extraction and relation extraction is that in a triple extraction process entities are not identified and they should be extracted from sentences along with relations between them. The main goal of triple extraction task is to convert unstructured text to a structured representation in the from of *<head, relation, tail>*. For this task, we developed a sequence to sequence model based on transformers, to generate the triples. The model encompass encoder and decoder which are initialized using publicly available checkpoints from other transformer models. We compared our model with other state-of-the-art models and showed that our approach achieves great results in generating triples from the input sequence.

Finally, we propose a procedure to create a knowledge graph from extracted triples. We take the extracted triples from the WebNLG dataset and build a weighted knowledge graph out of them.

Preface

This thesis is an original work by Mohammad Sahand Parniani. As detailed in the following, parts of the thesis have been published or submitted to journals or conferences in which Prof. Marek Reformat was the supervisory author and involved with concept formation and manuscript composition.

Abstract and Chapter 1, adopted some sections of the paper M. S. Parniani and M. Z. Reformat "Relation Extraction with Sentence Simplification Process and Entity Information," 2021 *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2021, pp. 635-640. A version of Chapter 3 has been published as M. S.Parniani and M. Z. Reformat, "Relation Extraction with Sentence Simplification Process and Entity Information," 2021 *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2021, pp. 635-640. A version of Chapter 4 has been submitted as a journal article to *IEEE Transactions on Knowledge and Data Engineering*.

"Life is like riding a bicycle. To keep your balance, you must keep moving."

- Albert Einestein

To my family and lovely wife

Acknowledgments

I would like to express my sincere thankfulness and appreciation to Prof. Marek Reformat for his infinite support and encouragement through this thesis. This research would not have been possible without his help and great supervision. I would like to extend special thanks to my examining committee members: Prof. Witold Pedrycz and Prof. Petr Musilek for taking time to review this research project.

Finally, I am thankful to my family who helped me through every moment of my life. My parents who raised me and supported me in different situation and got me to this point. Special thanks to my wife, who supported me through this whole thesis and inspired me.

Contents

List of Tables

List of Figures

Abbreviations & Acronyms

BERT Bidirectional Encoder Representation from Transformers.

BILSTM Bidirectional LSTM.

BOS Beginning of Sentence.

BPE Byte-Pair Encoding.

EOS End of Sentence.

FFN Feed Forward Networ.

GCN Graph Convolutional Network.

Glove Global Vectors for Word Representation.

GNN Graph Neural Network.

GTP Generative Pretrained Transformers.

IE Information Extraction.

KG Knowledge Graph.

LSTM Long Short Term Memory.

MLM Masked Language Modellin.

NER Name Entity Recognition.

NLP Natural Language Processin.

PCNN Piecewise Convolutions Neural Network.

RC Relation Classification.

RDF Resource Description Framework.

RE Relation Extraction.

RRN Recurrent Neural Networks.

T5 Text-to-Text-Transfer-Transformer.

TF-IDF Term Frequency - Inverse Document Frequency.

Chapter 1

Introduction

1.1 Motivation

The growing amount of information accessible to users generates several challenges related to its utilization. In most cases, the information is stored as text – sequences of words – that is impossible for machines to understand and take full advantage of it.

The introduction of the Semantic Web, particularly a graph-based data format, has enabled an attractive form of storing different types of information. The basic building block of this format is a triple. Each triple is made of a pair of entities – *head* and *tail*, connected together via a *relation* [1]. Large repositories of data represented as triples become new types of Knowledge Bases (KBs). Freebase [2], DBpedia [3] or Wikidata [4] are examples of well-known KBs that hold useful information represented in such a format. Sets of triples are highly interconnected, which allows for representing semantically rich information. Consequently, the information stored in KBs has been successfully used in multiple application domains: question-answering systems [5], recommendation systems [6], as well as knowledge discovery [7], to name just a few.

Despite many advantages, the construction and maintenance of KBs impose some challenges. For example, suppose a set of documents requires its 'translation' into triples to construct a new graph or update an existing one. Such processes – called in general relation extraction – are needed to build relevant KBs or keep the already

created graphs up to date. In most situations, such a process is performed manually, which is very costly and time-consuming. Therefore, a dedicated and automated process is necessary to extract relational facts – ⟨*head, relation, tail*⟩ – from the text and represent them in the structured format. The extracted facts are then added to KBs without human supervision. Therefore, it is imperative to have such processes in place to create Knowledge Graphs (KGs) automatically.

1.2 Objectives

This thesis focuses on designing a model/system to extract relational facts from plain text without human intervention. Processes of automatic extraction of knowledge facts can be perceived as having two activities. First, whenever new information is released, the unstructured information should be rapidly changed into a structured format <*head, relation, tail*> without human supervision. Second, the obtained structured information – triples – should be used to build new or update existing KGs. Such methods would ensure that KGs are always up to date and can be used in many applications.

The ultimate goal is to develop and validate a comprehensive system for extracting relational facts from plain text and use these extracted facts to build or update knowledge graphs (KGs). To accomplish the goal, the following objectives are identified.

1.2.1 Relation extraction (RE)

RE is a task to find a relation between a pair of entities already mentioned (identified) in a sentence. It can be modelled as a classification task that assigns a relation to a pair of entities mentioned in the text. The intent is to develop a novel neural-based model for relation extraction that uses information about entities and syntactic information from dependency trees as the input. We state that auxiliary information, like types of entities, can improve the RE performance based on performed experiments.

Also, dependency parsers that find syntactic dependencies between different tokens of a sentence should contribute to better performance. In general, rich structural information embedded in dependency trees has proven helpful for RE tasks. Such an approach has not gained much attention in the literature.

1.2.2 Triple Extraction

In contrast to RE, this process enables the extraction of triples from a text without entities being mentioned (identified/marked) in the sentences. In other words, the triple information extraction is the task of 'finding' triples: <*head, relation, tail*> in sentences. It is a domain-independent task where two entities and their corresponding relation are extracted simultaneously. Unlike in the case of RE, the process requires extraction of all possible entities, and it should be able to predict the most accurate relation between them. There could be more than one triple in a single sentence. The triples can share the same entity pairs. We aim to develop a transformer-based triple extraction model that can extract all types of triples in the sentence.

1.2.3 Constructing Knowledge Graph

Nowadays, Knowledge graphs(KGs) have attracted so much attention as a way to integrate data. KG is a great visualization tool that can be used to find a path between different entities and follow this path to find patterns between them. The graph-based database collects data from different types and sources and merges them through nodes and links. After merging all the data, we end up having a network of entities (nodes) that are linked together via relation(s). We plan to apply the triple extraction model to a text and generate multiple triples. Then, we aim to construct a knowledge graph via processing and connecting the triples.

1.3 Outline

The thesis is structured into 4 main chapters as follows:

In Chapter 2, we discuss all necessary background information that is required and the main ideas of this thesis are built on. Some of the information that will be discussed in this chapter are as follows:

- **Recurrent Neural Networks** that will be used for the proposed RE method.
- **Dependency Tree** that include useful syntactic information about the text.
- **Transformers** that have achieved state of the art on many natural language processing tasks.
- **Tokenization** that is necessary to tokenize the text and convert tokens to number.
- **RE** task and a reviewing some literature about that.
- **Triple Extraction** task and some related works about it.

Chapter 3 discusses the proposed RE method, which is a neural network model that is proposed to improve RE process. We develop a procedure using syntactic information obtained from dependency trees to remove noisy tokens from sentences and pick the most relevant ones that help extract correct relations. This method also utilizes entity type information, which is fed to the proposed model.

In Chapter 4, we will develop a triple extraction method. We treat triple extraction task, as a sequence to sequence task that the input is a input sentence that we want to extract all triples out of it and the output is a sequence of all possible triples in the input sequence.

In Chapter 5 we propose a method to build a knowledge graph from extracted triples. This will help us to apply queries on the triples in a nice visualise environment.

Finally, in Chapter 6, a summary of the significant results that are achieved in this thesis are listed and discussed.

Chapter 2

Background

In this chapter we provide a brief introduction for the concepts that this thesis is built on. In addition to that we investigate the recent approaches and methods, addressing relation extraction and triple extraction.

2.1 Recurrent Neural Networks

Text is a sequence of words which each word in this sequence relies on the previous words in the sequence. If an incomplete sentence is given to humans, probably they can complete the sentence based on the previous words given in the sentence, because their thoughts have persistence and use the information from the previous words in the sentence. However traditional neural networks assume all inputs and outputs are independent from each other and there is no connectivity between them. This is a major shortcoming for traditional neural models.

Recurrent Neural Networks (RNNs) address this issue by having a "memory" which allows them to capture the information from the current step of the sequence and pass it on through the next step [8]. Therefore the output of each step is dependent on the output of the previous step. Fig. 2.1 depicts RNN architecture. Given the sentence $X = \{x_1, x_2, ..., x_T\}$, each word is projected into embedding space $\{e_1, e_2, ..., e_T\}$, where $e_T \in \mathbb{R}^D$ and D and T are word vector dimension and the number of words in the sentence respectively. These embedding vectors are fed to the recurrent layer

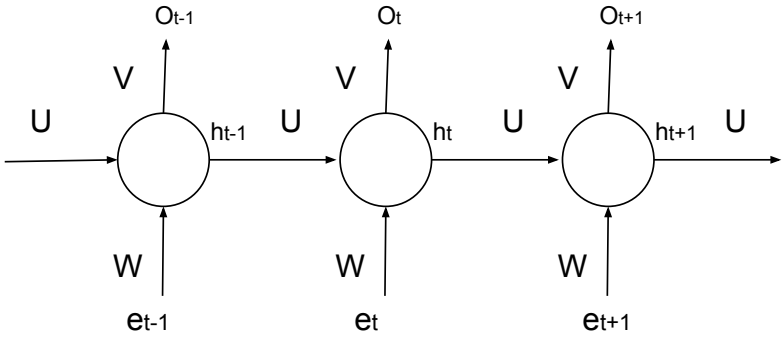

Figure 2.1: RNN structure

step by step. At the step t, RNN takes the embedding vector e_t and the output of the previous step h_{t-1}^{fw} as inputs and calculates the output for the current step h_t^{fw} [9]:

$$h_t^{fw} = tanh(W^{fw}.e_t + U^{fw}.h_{t-1}^{fw} + b^{fw}) \tag{2.1}$$

Where $h_t^{fw} \in \mathbb{R}^M$, $W^{fw} \in \mathbb{R}^{M \times D}$, $U^{fw} \in \mathbb{R}^{M \times M}$, $b^{fw} \in \mathbb{R}^M$ and M is the dimension of the feature vector. h_t^{fw} includes information for the sequence of words up to word $\{x_1, x_2, ..., x_t\}$. W^{fw}, U^{fw} and b^{fw} are the parameters of the model and $tanh(.)$ is the nonlinear transformation function. This function is usually $ReLU$ or tanh. All the steps in RNN share the same W, U and b. Therefore, for each step just the input changes. This reduces the number of parameters needed to learn significantly.

Fig. 2.1 depicts the RNNs. This diagram has output for each time step which is not necessary for all the tasks. For example if we want to do text classification, we just need the output of the last step for classification.

2.1.1 Bi-directional RNNs

The main drawback of one directional RNNs is that we cannot utilize the information from future words and use them to predict a word in the middle of the sequence. Utilizing both past and future words for predicting the semantic meaning in the middle of a sentence can boost the performance. For example when we are asked to predict a missing word in the sentence, we may take a look at both left and right

words of the missing words to predict better. The solution is using bi-directional architecture which utilizes both past and future words to make predictions. The output at the time step t during the backward processing is obtained by:

$$h_t^{bw} = tanh(W^{bw}.e_t + U^{bw}.h_{t-1}^{bw} + b^{bw}) \tag{2.2}$$

Where h_t^{bw}, U^{bw}, W^{bw} and b^{bw} have the same dimension as h_t^{fw}, U^{fw}, W^{fw} and b^{fw} respectively. Backward and forward RNNs are trained simultaneously during the training phase. With the bidirectional RNN architecture, the output at the time step t is obtained by concatenating h_t^{bw} and h_t^{fw} together.

2.1.2 Long Short Term Memory Networks

The main idea behind RNNs architecture is that they may be able to connect information from the previous steps to the current step. For example, for predicting the next word in a sentence, we may need the words coming before the missing word. But what if for predicting the missing word, we need the context which is so further back. For instance, in the sentence *"I was born in France, which is a very beautiful country and that's why I can speak (A) fluently."* we know the word A is a language but for predicting it we need the word France from the sentence which there is a large gap between them. Unfortunately RNNs are not capable of handling long term dependencies and they suffer from vanishing gradient problem [10]. For solving this issue Long Short Term Memory(LSTM) networks are introduced [11] which are a special kind of RNNs.

LSTMs are designed to solve long term dependency problem and therefore can remember information for a long period of time. The main idea is to provide a gating mechanism for each unit, which decides to keep the information from previous states or not. LSTMs include 4 components: an input gate (i_t), a forget gate (f_t), an output gate (o_t) and a memory cell (c_t). These components are shown in Fig. 2.2. We can

write equations for these components as below [12]:

$$i_t = \sigma(W_i.e_t + U^i.h_{t-1} + b^i)) \tag{2.3}$$

$$f_t = \sigma(W_f.e_t + U^f.h_{t-1} + b^f) \tag{2.4}$$

$$o_t = \sigma(W_o.e_t + U^o.h_{t-1} + b^o) \tag{2.5}$$

$$c_t = i_t \otimes g_t + f_t \otimes c_{t-1} \tag{2.6}$$

$$g_t = tanh(W_g.e_t + U^g.h_{t-1} + b^g) \tag{2.7}$$

Here g_t extracts the feature vector for the current step in LSTM. c_t is the memory cell state for the current step and is computed based on the previous cell state and g_t. The hidden state computed for each LSTM unit equals to:

$$h_t = o_t \otimes tanh(c_t) \tag{2.8}$$

In all of the equations mentioned above, $\otimes$ is the sign for element-wise multiplication and σ is the sigmoid transformation function. As we can observe from the equations above, memory cell state is updated in each step and LSTMs can learn to forget previous states information or just keep them and pass it to the next step.

2.2 Dependency Parsing

Dependency parsing is the task of analyzing the grammatical structure of the sentence and building relation among the "head" word of the sentence and other words modifying it [13]. Parsing resolves the structural ambiguity of the sentence in a form. Fig. 2.3, depicts typed dependency structure for the sentence *"I take the bus every day to school"*. An arrow from the "take" towards "I" indicates that the word "I" is modifying the word "take". Each of the arrows in the Fig. 2.3 has a label which indicates the relationship between words.

Dependency parser can simply transfer a sentence into the dependency tree. Dependency tree is a structure in a graph format with $|V|$ nodes and $|A|$ links. V is

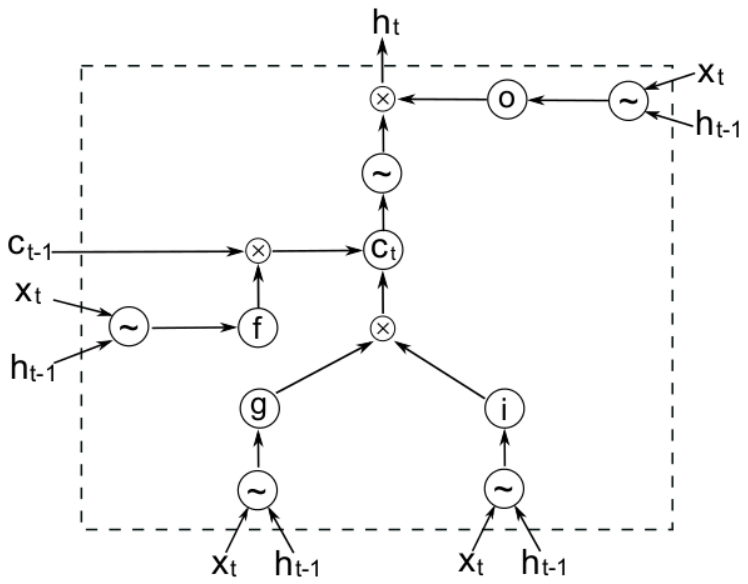

Figure 2.2: LSTM unit. h: hidden unit. c: memory cell. i: input gate. f : forget gate. o: output gate. g: candidate cell. $\otimes$:element-wise multiplication. $\sim$: activation function.

the number of the words in the sentence which are connected through the links. A describes the dependency relation between words of the sentence. For example in the link $h \rightarrow d$, h is the head word and d is the dependent. Root is the head for all other nodes in the tree (directly or indirectly) and therefore is the most important node in the tree. Fig. 2.4 depicts the dependency tree structure for the mentioned sentence above.

2.3 Precision-Recall Curve

Precision can be defined as the number of true positive (T_p) samples divided by the summation of true positive and false positive (F_p) samples. However, recall can be defined as the number of true positive samples divided by the summation of true

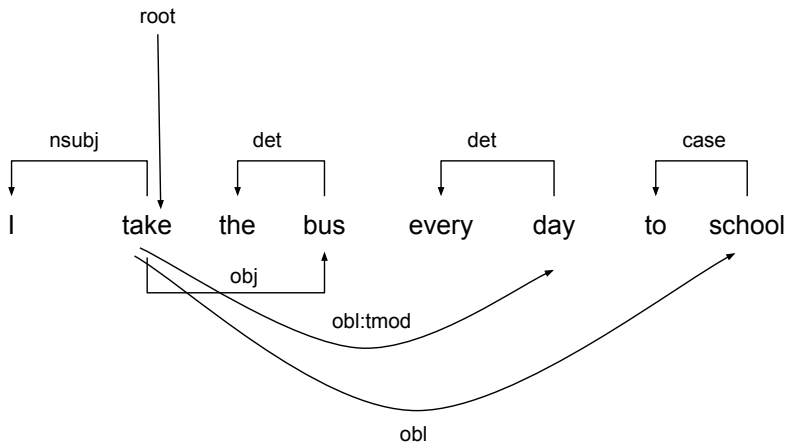

Figure 2.3: Typed dependency structure for the sentence *"I take the bus every day to school"*

positive and false negative (F_n) samples.

$$P(Precision) = \frac{T_P}{T_P + F_P} \tag{2.9}$$

$$R(Recall) = \frac{T_P}{T_P + F_N} \tag{2.10}$$

Precision-Recall curve is a metric to measure how successful our classifier was. This metric is extremely useful when our dataset is highly imbalanced. This curve plots precision and recall when the threshold changes from 1 to 0. When precision and recall are both high, it means the area under the curve is high too. Higher precision means few numbers of false positive samples and high recall means few numbers of false negative samples.

2.4 Word Embedding

Machine learning algorithms and deep learning architectures are not able to process raw string or plain text. Therefore, texts should be converted into real vector numbers to be understood for neural networks as an input feature. Word embedding is the process of converting each word in the document into a vector using a dictionary. For example in the sentence *"I go to school by bus"*, the easiest way to represent words in using one hot vector for each word in the sentence. In this case our dictionary is

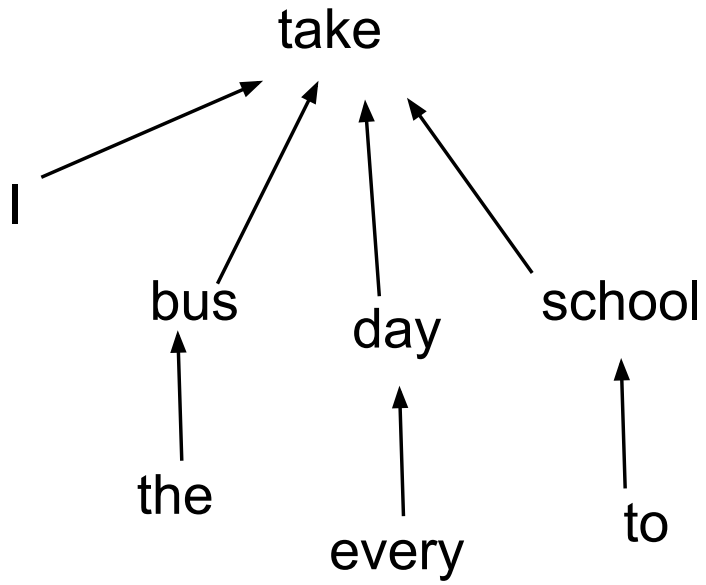

Figure 2.4: Dependency tree structure for the sentence *"I take the bus every day to school"*

['I', 'go', 'to', 'school', 'by', 'bus'] and then the vector representation for the word *"I"* is $[1, 0, 0, 0, 0, 0]$ and etc. But this is the simplest form we can represent words as vectors. There are different methods for word embedding which we mention some of them in this section.

2.4.1 Count Vector

consider a corpus D which includes N documents. In this corpus we can extract all unique tokens and form a dictionary. Let's say we have T unique tokens. Then we can count the frequency of each token for each document and build a count matrix M, which is in the size of $N \times T$. Each column of M, represents the word embedding for different tokens and each row in M represents the frequency of different tokens in each document. For instance let's say we have the following corpus:

D=["This is the first document",
"This is the second document",
"This is the third one"]

There are 3 documents in D and our dictionary will be *['This', 'is', 'the', 'first', 'document', 'second','third', 'one']*. The matrix M is the size of 3×8 which is shown below:

Table 2.1: Word count for the document D

	$This$	is	the	$first$	$document$	$second$	$third$	one
D_1	1	1	1	1	1	0	0	0
D_2	1	1	1	0	1	1	0	0
D_3	1	1	1	0	0	0	1	1

The first row of M represents the frequency of each word in the dictionary in the first document and the first column of M represents the vector representation obtained for the word "This". Using this method may lead to a sparse matrix for M specially if we have a large corpus and therefore many unique tokens. In this case we will have lots of zeros in M and is not efficient for computation.

2.4.2 TF-IDF Vectorization

The main idea behind this method is applying higher weight to more important tokens in the corpus and lower weight to less important one. If we have a large corpus, words like "the", "is", "a" appear in most of the documents. So, these words are not suitable for classifying documents. In the count vector method, these words are not sparse, claiming they carry lots of information. In the TF-IDF method, we want to lower the importance of these tokens by applying a weight to them:

$$TF - IDF(term, document) = TF(term, document) * IDF(term) \tag{2.11}$$

The first term is term frequency (TF) which indicates the occurrence frequency of each term in the document. This means the number of times each word occurs in the document divided by the number of words in the document and is obtained as follow:

$$TF(term, document) = \frac{n_i}{\sum_{k=1}^{V} n_k} \tag{2.12}$$

The second term is inverse document frequency (IDF) and indicates the inverse of the number of documents that each term appears in. This is obtained as follow:

$$IDF(term) = \log \frac{N}{n_t} \tag{2.13}$$

Take a look at the corpus D shown in the previous section. We can calculate TF-IDF for the two words *"this"* and *"first"* in the first document as follow:

$$TF - IDF(\text{"}this\text{"}, document_1) = \frac{1}{5} \times \log \frac{3}{3} = 0$$
$$TF - IDF(\text{"}first\text{"}, document_1) = \frac{1}{5} \times \log \frac{3}{1} = 0.0954$$

As we can see, this method penalizes the word "this" significantly as it appears in all the documents and so does not carry too much information. But we assign higher weight to the word "first" because it just appears in the first document. Now we can get to the word vector representation by replacing elements of matrix M from the previous section with their respective TF-IDF elements.

2.4.3 Word2Vec

Word2vec [14] is the first neural embedding model. This method is a predictive based model which calculates probability for each token. This method was also the first method to achieve the task Germany-Berlin=France-Paris, which was a big move for embedding methods. Word2vec is a combination of two different techniques: CBOW(continuous bag of words) and skip-gram model. Both techniques use shallow neural networks which map word(s) into word(s). Both techniques learn weights which can be interpreted as word vector representations. Both methods are explained in the following:

CBOW:

CBOW predicts the probability of a word given a context. Context can be a single word or multiple words. Let's say the context in a single word for simplicity. The

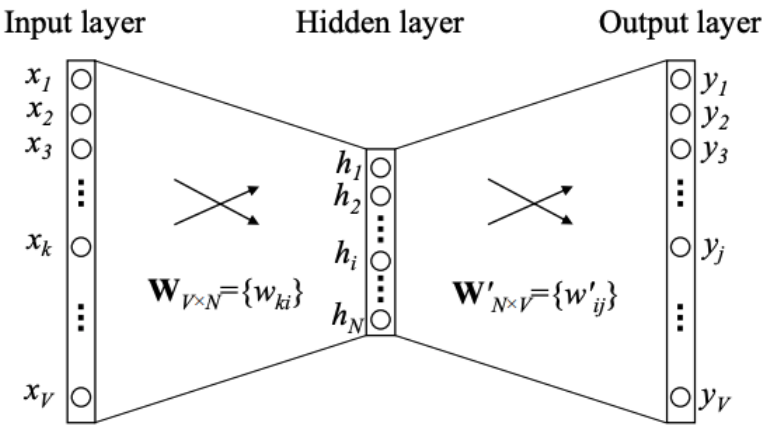

Figure 2.5: Shallow neural network used in CBOW (single word context) [15]

input is the one-hot encoder of the context which is in size of $1 \times V$ and V is the vocabulary size. This input is fed to a shallow neural network (Fig. 2.5) which has just one hidden layer. Input weight is in size of $W = V \times N$. The output vector is also in size of 1*V and output weight is in size of $W^{'} = N \times V$. N is a hyperparameter which we use to choose the dimension of word vector representation. After training the neural network, we take W and $W^{'}$ as a word vector representation.

Skip-gram Model:

Skip-gram model has the same architecture as CBOW but it has been flipped. Skip-gram predicts the probability of context given a word as input. Simply, we want to predict the context around, given the center word. Input is a one hot encoder vector of size $1 \times V$. Hidden layer has N neurons which is the size of word vector representation. Output is in size of $C[1 \times V]$ where C is the context size. Fig. 2.6 Depicts skip-gram model.

2.4.4 Glove

After [14] released their word2vec method, many articles were published about word embedding. One of the best articles in this area was Glove (Global Vectors for

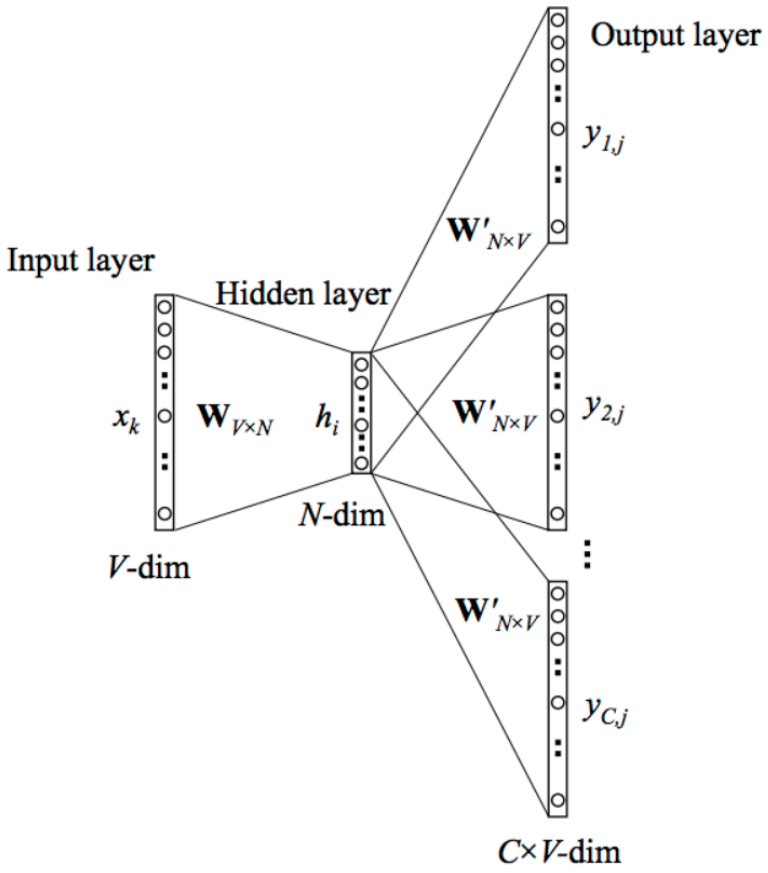

Figure 2.6: Shallow neural network used in Skip-gram [15]

Word Representation)[16] which reformulated word2vec algorithm as a special kind of factorization for word co-occurrence matrix. Glove tries to capture word embedding by observing the whole corpus. Word frequency and co-occurrence counts are the main things that Glove makes use of them. Glove form co-occurrence matrix X which element X_{ij} in this matrix indicates the number of times word j has been in the context of word i. As a result:

$$P_{ij} = P(j|i) = \frac{X_{ij}}{X_i} \tag{2.14}$$

Where P_{ij} is the probability that word j occurs in the context of word i. The loss function that Glove minimizes it is:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \check{w}_j + b_i + b_j - \log X_{ij})^2 \tag{2.15}$$

Where f is a weighting function which is created manually and W_i and W_j are word vectors for words i and j respectively. Glove is trained on a global co-occurrence matrix to minimize the loss function by minimizing the least square error and produce word embedding vectors in meaningful space.

2.5 Transformers

Nowadays, Transformers [17] have become popular in different artificial intelligence fields such as: Natural Language Processing (NLP), Computer Vision (CV) and Speech Processing. The first Transformer was proposed in [17] which was utilized towards a machine translation model. After that, researchers came up with the idea of using the vanilla model as an initial design and adopt it through different architectures, which resulted in state-of-the-art for different NLP tasks. Some of the new models which are based on the vanilla architecture are BERT[18], GPT[19] and RoBERTa[20]. These models are pretrained on large datasets and then fine-tuned on the downstream tasks. In the following sections, we first describe the vanilla transformer architecture and then investigate different variants of it.

2.5.1 Vanilla Transformer

Vanilla Transformer is a sequence to sequence model which was introduced for machine translation [17]. This sequence to sequence model is made up of an encoder and a decoder, each consisting of identical blocks. Each encoder block is made up of a bi-directional Multi-Head-Self-Attention module and a position-wise feed forward network (FFN). For making a deeper model, a residual connection [21] and a normalization layer [22] are added around each module. This proved to improving the

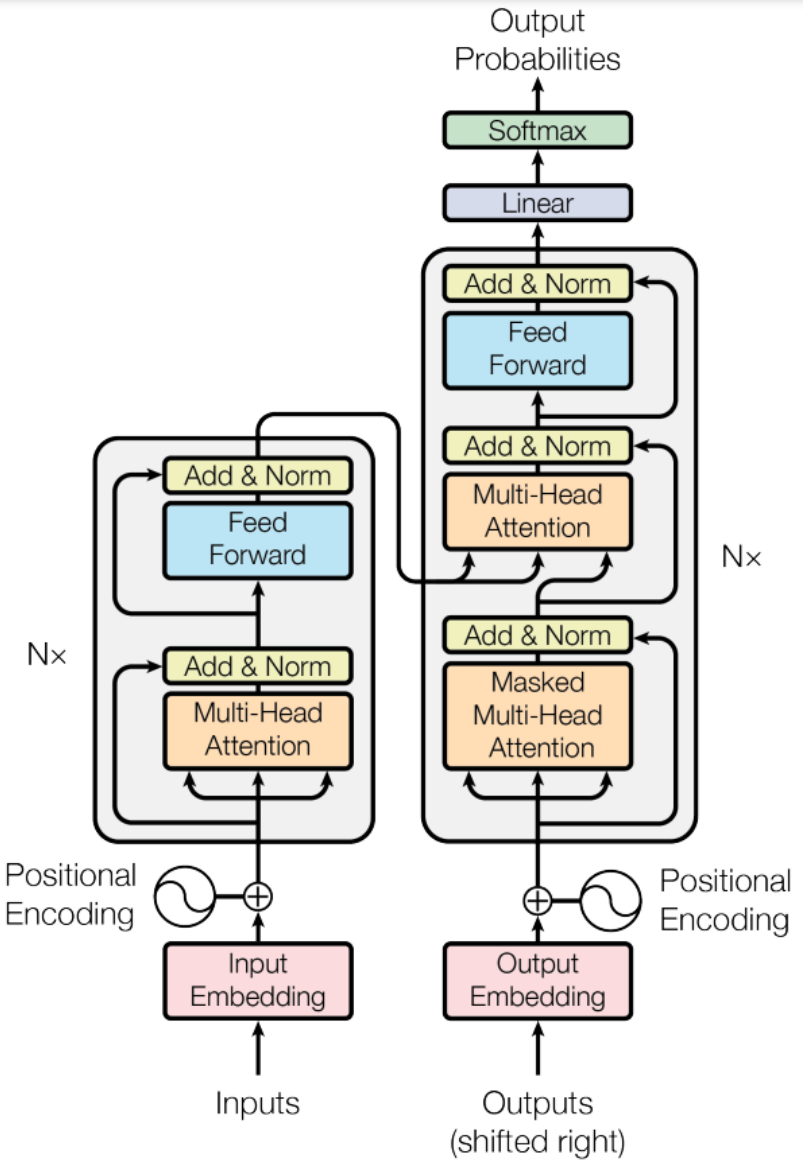

Figure 2.7: Structure of a vanilla encoder-decoder transformer [17]

performance of the model.

Structure of the decoder is the same as the encoder with some minor changes. The Multi-Head-Self-Attention block in the decoder is unidirectional, which prevents each position from attending subsequent positions for calculating the attention score. In addition, the decoder block has a cross attention layer between the outputs of the encoder and the output of the decoder's self-attention block. Fig. 2.7 depicts the architecture of a vanilla transformer. Vanilla Transformer consists of 6 stacked

encoder block and 6 stacked decoder block.

Different modules that are used in the transformer's architecture is described in the following sections:

Attention Modules

When an input sequence is thrown to the transformer, the self-attention module tries to extract the relationship between different words (positions) in the sequence. To calculate the self-attention, three vectors: query(Q), key(K) and value(V) should be created for the input sequence. These vectors are created by multiplying the word embedding matrix by query (WQ), key(WK) and value(WV) matrices respectively. These matrices are optimized toward optimal values during the training process.

After obtaining $Q \in \mathbb{R}^{(N \times d_k)}$, $K \in \mathbb{R}^{(M \times d_k)}$ and $V \in \mathbb{R}^{(M \times d_v)}$ for the input sequence, the next step is calculating attention scores as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.16}$$

Where d_k, M and N represent the length of the query matrix, dimension of the key matrix and dimension of the query matrix respectively. As it can be seen from Equation (2.16), the dot product of Q and K^T is divided by $\sqrt{d_k}$, to prevent gradient vanishing problem which results in poor performance.

To improve the performance of self attention layer, [17] proposed using multi-head-attention design which has some advantages over single-head-attention one:

1. it helps the model to focus on different positions of the input sequence. In this case, all the input tokens collaborate to the final representation, which prevents it from being dominated by the words in a specific position

2. There will be multiple sets of query, key and value matrices (one for each attention head) which are used to project the embedding matrix into different representation subspaces which improves the attention functionality.

For each attention head, the attention is calculated using Equation (2.16) and then all outputs are concatenated back together to build the final attention matrix.

$$Multi\_Head\_Attention(Q, K, V) = concat(head_1, \cdots, head_n) \tag{2.17}$$

$$head_j = Attention(Q_j^Q, K_j^k, V_j^V) \tag{2.18}$$

Vanilla Transformer has 8 attention heads. hence, n in Equation (2.17) is equal to 8.

Three different attention mechanisms that are applied in transformer structure are as follows:

1. Self_Attention: Which is used in the encoder. In this type, each query is allowed to attend keys from all positions in the input sequence. So, forward and backward tokens participate in the attention calculations.

2. Masked_Self_Attention: Which is used in the decoder. In this type, each query is just allowed to attend previous positions and itself. To make this possible, an upper triangular matrix (Mask matrix) should be created which its size is equal to the length of the output sequence. Matrix's elements are filled as follow:

$$\begin{gathered} M_{ij} = -\infty \; if \; i < j \; , \\ else \; M_{ij} = 0 \end{gathered} \tag{2.19}$$

   Where M is the mask matrix. This type of attention can be called with different names such as: casual attention, auto-regressive attention or unidirectional attention.

3. Cross attention: which is used in the decoder. This layer is located after the masked-self-attention layer and calculates the attention between the outputs of the encoder (keys) and the outputs of the masked-self-attention layer (queries).

Position-wise Feed Forward Network

Position-wise Feed Forward Network (FFN) (in which its parameters are shared among different positions) is a fully connected feed forward network. It's called position-wise because the same network would be applied to each position of the sequence:

$$FFN(H^{'}) = ReLU(H^{'}W_1 + c_1)W_2 + c_2 \tag{2.20}$$

In Equation (2.20), $H^{'}$ is the output of the previous layer and $W_1 \in \mathbb{R}^{(D_m \times D_f)}$, $W_2 \in \mathbb{R}^{(D_f \times D_m)}$, $c_1 \in \mathbb{R}^{D_f}$ and $c_2 \in \mathbb{R}^{D_m}$.

Residual and Normalization Module

If we take the self attention and position-wise FFN layers as a combined package, connection[21] and normalization[22] module are connected to each package which allows gradients to pass through the model easier. This will reduce the training time significantly.

$$H^{'} = LayerNorm(SelfAttention(X) + X) \tag{2.21}$$

$$H = LayerNorm(FFN(H^{'}) + H^{'}) \tag{2.22}$$

Position Encoding

When the input sequence is fed to the transformer, since there is no recurrence in the architecture of Transformer, order of the words might be missing. To prevent this, a positional embedding vector is added to each word's embedding vector. The idea behind this is tracking the positions of different tokens such that there will be a meaningful distance between different words in the sequence, when their embedding is projected into Q, K and V matrices.

In the vanilla Transformer, positional information vector is obtained using absolute sinusoidal position encoding. For example, for obtaining the position vector at the position t, the following formula is used:

$$PE(t)^i = \begin{cases} sin(\omega t_i) & \text{if } i \text{ is even} \\ cos(\omega t_i) & \text{if } i \text{ is odd} \end{cases} \tag{2.23}$$

Where i varies from 0 to D_m. ω is the frequency of the sinusoidal function and is pre defined.

2.5.2 Different Transformer Architecture

There are three different variant, that transformers can be used:

1. Encoder-Only Transformer: where there is no decoder and we just have the encoder in the architecture. Outputs of the encoder can be used as a representation for each word in the input sequence. This model is useful in sequence classification or word labelling tasks such as Name Entity Recognition.

2. Decoder-Only Transformer: where there is no encoder in the architecture. Since there is no encoder, the cross attention layer is withdrawn from the decoder. This variant is good for sequence generation tasks.

3. Encoder - Decoder Transformer: Both encoder and decoder exist in the architecture like vanilla Transformer. This model is good for sequence to sequence tasks such as machine translation or text summarization tasks.

We will mention and discuss some variants of Transformers in the following parts:

BERT

BERT(Bidirectional Encoder Representation from Transformers)[18] is an encoder-only transformer which achieved state-of-the-art results for some NLP tasks such as: Question Answering (SQuAD v1.1) and Natural Language Inference (MNLI).

BERT's game changing idea was proposing the Masked Language Modelling (MLM) method, which enabled BERT to train transformers for language modeling in a bidirectional manner. Before BERT, transformers were trained in a single direction (from left to right or from right to left). Training Transformers in a bidirectional manner, helps the model to have a deeper and better understanding of the text. Having this, the model learns the representation of each word using all its surrounding context (both left and right context) and can perform better on downstream tasks.

The main challenge of the BERT is how to make bidirectionality possible during the training. Usually other methods train their language models through the next word prediction task. However, the next word prediction task limits the ability of the model to have access to all the surrounding text around the word and just have access to the previous context. To avoid this problem, BERT utilizes the MLM method.

MLM is a crucial part in BERT architecture. When feeding the input sequence to the BERT, %15 of tokens are randomly masked. When a token is masked, it is replaced with the token $[MASK]$. Now the model should try to predict these masked tokens and find out their original values. Hence, we should add a classifier and softmax layer on top of the BERT's output vectors. It should be noted since BERT just predicts the masked tokens, thus just the prediction of masked tokens in the output are considered in the loss function and prediction of non-masked tokens are ignored.

Another task that BERT performs during the training is the Next Sentence Prediction (NSP) task. This part helps the BERT to achieve better performance for sequence to sequence tasks. The input sequence feeding to the BERT, is a pair of sentences which we can call sentence A and sentence B. During the training phase, BERT predicts if the sentence B is the subsequent sentence of A in the original document or not, based on the meaning. During the training 50 percent of times, B is a subsequent sentence of A and for the other 50 percent, B is just a random sentence chosen from the corpus.

The input sequence <*"my dog is cute. He likes playing"*> feeding to the BERT is shown in the fig. 2.8. Each input token is converted to a vector by summing its position embedding vector, segment embedding vector and token embedding vector. Segment embedding vector demonstrates if the token is in sentence A or sentence B. Also, $[CLS]$ token is added to the beginning of the sequence and $[SEP]$ token is added to the end of each sentence. The output representation of $[CLS]$ token can be used as a sentence embedding vector that embeds the representation of the whole sequence. $[CLS]$ token is used in the training phase to perform the NSP task.

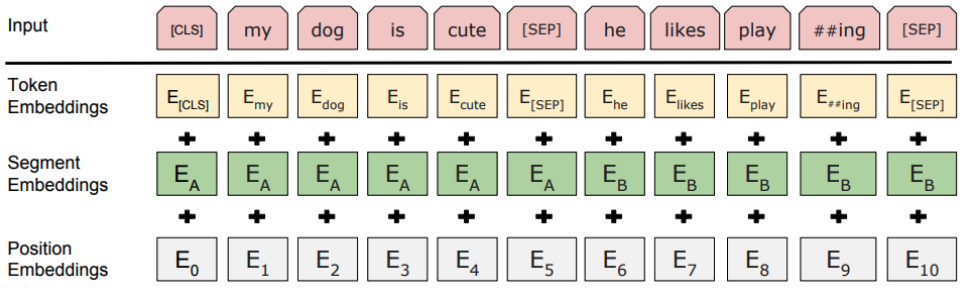

Figure 2.8: How different vectors are concatenated to build the input of the BERT model [18]

Fine-tuning BERT on Downstream task: When BERT is pretrained on large corpus using two discussed strategy: MLM and NSP, the next step is fine-tuning the pretrained BERT on downstream NLP tasks. In this section we briefly discuss how some NLP tasks are fine-tuned using BERT:

1. *Sequence classification* tasks such as sentiment analysis can be done by adding a classification layer on top of the $[CLS]$ token representation.

2. *Name Entity Recognition (NER)* task can be done by adding a classification layer on top of each token's output representation. Hence, each token is classified as one of the existing name entity tags.

3. For *question answering* task, the model has to learn two vectors that specify the beginning and end of the answer in the document.

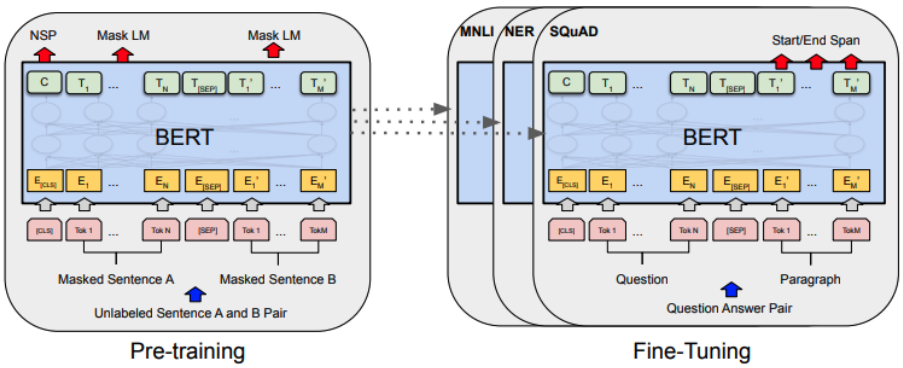

Figure 2.9: Pretraining BERT using MLM and NSP and fine-tuning on the downstream tasks [18]

BERT parameters: Two different BERT models were explained in [18]: Base model and Large model. Base model has 12 stacking encoder blocks, 12 attention heads and hidden size of 768, which in total results in 110M parameters. On the other hand, Large BERT has 24 stacking encoder blocks, 16 attention heads and hidden size of 1024, which in total has 340M parameters.

Generative Pretrained Transformers

Generative Pretrained Transformers(GPTs) are decoder-only transformers that are pretrained in unsupervised manner for language modeling. GPTs are used for sequence generation tasks in NLP such as text summarization and question answering. There are three different GPTs released so far which we discuss in this section:

GPT-1: GPT-1 was proposed in[19]. In this paper, a generative language model was proposed that is trained using unlabeled data (unsupervised manner). After training the language model, we can fine-tune it on downstream tasks providing few supervised examples. Hence, this approach is a semi-supervised learning method (unsupervised language model pretraining and supervised downstream task fine-tuning).

1. Unsupervised Language Modelling: Which is used in pretraining process and its objective is as follows:

$$L_1(J) = \sum_i \log P(j_i | j_{i-k}, \cdots, j_{i-1}; \theta) \tag{2.24}$$

Where j_i represents the token at position i and θ is the set of all parameters of the model.

2. Supervised fine-tuning: In this part, the model is fine-tuned on downstream tasks in a supervised manner. Therefore the objective formula can be written as follows:

$$L_2(D) = \sum_{x,y} \log(y | x_1, \cdots, x_n) \tag{2.25}$$

Where D is the labelled dataset, y is the labeled output and X is the set of all features. In[19], it is recommended to use an auxiliary learning objective which is written as follows:

$$L_3(D) = L_2(D) + \lambda L_1(D) \tag{2.26}$$

Where $L_1(D)$ is the learning objective related to the language model and λ is a weight given to this objective and equals to 0.5. For fine-tuning GPT on downstream tasks, a linear and softmax layer should be added on the top of the transformer's output.

3. Input Format for Each Task: In order to make the input sequence ready for the GPT, some changes are done on the sequence. Such as: start and end tokens are added to the sequences and also a special (delimiter) token is added between different parts of the training example to make sure that the input is fed to the model in an ordered manner.

GPT-1 is pretrained on the Books-Corpus dataset. It has 12 stacked decoder blocks with 12 attention heads. The hidden size of the model is 768 and it has total parameters of 117M.

GPT-2: After GPT-1, GPT-2[23] was released. GPT-2 was pretrained on a larger dataset compared to GPT-1 and had more parameters such that it has a stronger language model. There are two important concepts relevant to GPT-2:

1. **Task Conditioning**: Training objective for language models can be formulated as maximizing the probability, $p(output/Input)$ which means maximizing the probability of the desired output, given the input dataset. However, GPT-2 tries to learn multiple tasks. So, the training objective for GPT-2 can be written as $p(output/Input, task)$, so the task is also added to the optimizing process. This implies that our language model should generate different outputs for the same input sequence but related to different tasks. This characteristic is known as task conditioning. In language models, task conditioning is conducted by providing a specific instruction about each task to the model.

2. **Zero Shot Transfer Learning**: One of the main characteristics of GPT-2 is its capability to perform a zero shot task transfer. In zero shot task transfer, the model understands the task without any given example and just by providing an instruction about that specific task. For example if we want the model to do an English-Chinese translation task, we can just pass the English sentence to the model followed by "Chinese:" word and therefore the model understands this is a translation task and performs the task.

GPT-2 dataset: Dataset used for training GPT-2 is WebText which has around 40GB of text. Only high quality text and articles are included in WebText and low-referenced articles are removed. Also, all Wikipedia articles are removed from the

dataset as test set contains many Wikipedia articles. Compared to the Book Corpus which GPT-1 was trained on, this dataset is much larger.

GPT-2 Architecture: GPT-2 has 48 stacked decoder blocks and the size of its word embedding is 1600. It has a larger vocabulary size compared to GPT-1 which equals to 50257. The length of the input can be as long as 1024. The position of the normalization layer is changed in GPT-2 and is moved to the input of each sub-block.

GPT-2 was trained in 4 different cases (different model sizes), 117M parameters, 345M, 762M and 1.5B. Results demonstrate that model performance gets improved by increasing the number of parameters.

GPT-2 proved that increasing the capacity of the model and training on larger dataset, can increase the performance of the model significantly in zero shot setting and reduce perplexity. Another interesting observation was that GPT-2 under-fitted the WebText dataset, which means by increasing the capacity of the model even further, the performance improves more and perplexity decreases further.

GPT-3: GPT-3 was proposed in[24]. GPT-3 has 175B parameters which is 10 times larger than GPT-2. in addition to the the larger model, GPT-3 was pretrained on a larger dataset. This huge capacity and the large dataset that GPT-3 was trained on, makes it a very strong and powerful language model, which performs well on downstream tasks in zero shot settings. Due to GPT-3's large capacity, it can perform well on tasks which it wasn't trained on such as writing different programming codes like python, just by providing natural language description of the task.

Learning Objectives for GPT-3: There are two objectives, that GPT-3 is trained based on them:

1. **In context learning**: GPT-3 was trained on a very large dataset. During the training phase, GPT-3 is learned to predict the next word in the sentence.

However, it will start learning about the overall structure of the text and its different patterns. This helps the model to perform well on zero task transfer, when a few examples or a description about the task is provided. What the model does is to compare the given examples with what it has learnt from the past and so find a similar pattern between them. It is clear that when the capacity of the model increases, the model's ability to perform zero setting learning increases.

2. **Zero shot, Few shot and One shot setting**: Zero shot, few shot and one shot settings are a subset of zero task transfer. In the zero shot setting, the model is provided with no example and just a description about the task, in one shot setting, the model is provided with an instruction and one example about the task. In the few shot setting, the model is provided by an instruction and so many examples as it fits into the context window size of the model.

Dataset: GPT-3 was trained on 5 different datasets. These datasets are Books1, Books2, Wikipedia, Common Crawl and WebText2. Datasets that have higher quality and more trustable, are given a higher rate in the training such that the model is trained on them for more than one epoch.

GPT-3 Architecture: GPT-3 has 96 layers with 96 attention heads in each layer. Word embedding dimension is 12888 which is around 8 times more than GPT-2. Also, the size of the context window is increased to 2048. In GPT-3 locally banded sparse attention is used which improve the performance of the model.

GPT-3 was evaluated on some language modeling dataset and it achieved better results on LAMBADA and Penn Tree Bank datasets in both few shot and zero shot settings. In addition, it achieved state of the art results on some NLP tasks like question answering and translation. However, the model's performance is better in few shot setting compared to zero or no shot setting in most of the cases.

RoBERTa: RoBERTa was proposed in [20] and is a short term for Robustly Optimised BERT Pre Training Approach. RoBERTa has relatively the same architecture as BERT. The main goal of RoBERTa was optimizing the training process of BERT in order to achieve better results on downstream tasks. Some modifications that are performed on BERT are mentioned below:

1. **Discarding Next Sentence Prediction (NSP) part**: In BERT, one of the objectives in pretraining is NSP. Any input sequence is a pair of two sentences which BERT tries to predict if the second sentence is the subsequent of the first sentence in the original document or not. In RoBERTa NSP loss is removed from the model loss and they noticed that by doing this, they achieve better results on downstream tasks.

2. **Increase the batch size in pretraining**: BERT is trained for 1M steps with a batch size of 256. On the other hand, RoBERTa is trained for 125 steps with a batch size of 2k and 31k steps with a batch size of 8k. Larger batch size decreases the perplexity of the language model and therefore achieves better performance on downstream tasks. The other good thing about batch size is making parallel training easier.

3. **Changing masked words for different epochs**: In BERT, masked words are assigned once and randomly and they stay the same during the whole training phase. However in RoBERTa, training data is trained for 40 epochs and each 4 epochs, a different masking strategy is utilized. This can be called dynamic masking.

Dataset: For training RoBERTa, 4 different datasets are used. BOOK CORPUS and English Wikipedia dataset (16GB of text), CC-NEWS (76GB of text), OPEN WEBTEXT (38GB of text) and STORIES (31GB of text). Among these, BOOK CORPUS and English Wikipedia dataset were used for BERT training as well.

RoBERTa's Performance: Following are the results of applying RoBERTa on different datasets:

1. RoBERTa achieved better results on the RACE dataset compared to BERT large and XLNet[25].

2. RoBERTa achieved state of the art results on 4 GLUE datasets: MULTI Natural Language Inference (MNLI), Semantic Textual Similarity (STS), QuestionNLI and Recognize Textual Entailment (RTE).

3. RoBERTa achieved the same results as XLNet on SQUAD 1.0 and SQUAD 2.0 datasets.

T5 Model: T5 or Text-to-Text-Transfer-Transformer was proposed in [26]. In T5, a large empirical study is performed to compare different transfer learning methods that were proposed up to that date and based on that propose a new model that is called Text-to-Text-Transfer-Transformer or T5. In addition to that, a new dataset was released which is called Colossal Clean Crawled Corpus (C4) dataset. T5 was pretrained on this dataset and achieved state-of-the-art results on most of the NLP benchmarks.

In T5, all NLP tasks are converted into a text to text problem where input and output are both text strings. This is different from BERT where the output could be just a class label or a span of the input text. T5 is a model that combines all NLP tasks (including machine translation, sentiment analysis, question answering and summarization) into a unified model, which all use the same loss function and hyperparameters. The output of T5 is a text, however we can output a number as well in T5. This can be done by converting the output number into a string text. Therefore, T5 can be also used for regression problems. Fig. 2.10 depicts the proposed framework of T5 model.

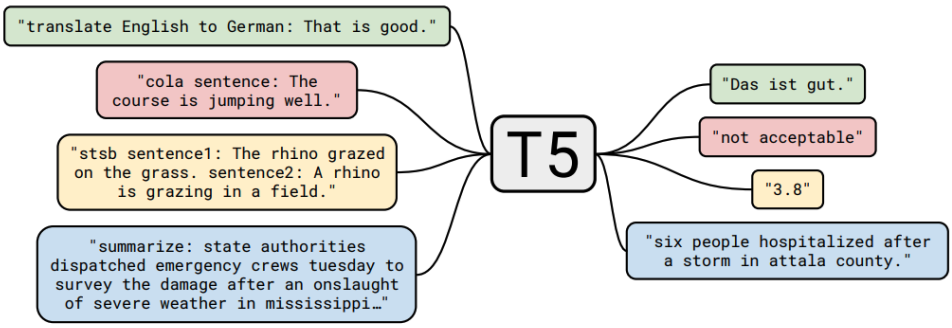

Figure 2.10: T5 framework where multiple NLP tasks are combined into one model [26]

Dataset: One of the main important steps in transfer learning is pretraining on the unlabelled dataset. To increase the performance of the model on downstream tasks, pre-training dataset should be large, clean and diverse. For example, the Wikipedia dataset is clean, but not diverse and large. On the other hand, the Common Crawl web scareps dataset is large and diverse, but not clean. To address these issues, T5 was pre trained on the Colossal Clean Crawled Corpus (C4) dataset which is a clean version of the Common Crawl web scareps dataset. In the cleaning process, duplicate sentences are removed, incomplete sentences are removed and also offensive sentences are removed. This will give us a clean, diverse and large dataset which is larger than Wikipedia dataset.

Studying previous transfer learning methods: When the T5 model is created and pretrained on the C4 dataset, the authors investigate previous transfer learning methods. They conducted experiments on the followings:

1. **pre training objective**: In BERT, MLM was proposed where the model was supposed to find the masked word, however in other language models, next word prediction was used as a self-supervised task. Authors in T5 noticed that if the objective is recovering the missing (masked) words, the model achieves better results.

2. **Model Architecture**: It was found that models with encoder-decoder architecture achieve better results compared to decoder-only transformers.

3. **Pre Training dataset**: It was found that if the unlabelled dataset used for pretraining is small, it can lead to over-fitting which is not good. On the other hand, training on the in-domain dataset can be good for the model's performance.

4. **Training strategies**: Authors in [26] found out that multitask learning strategy can lead to competitive performance compared to pretraining and then fine tuning learning strategy. However, when using multitask strategy, we should be careful about how much time we should train the model for each task.

5. **Model Size**: In [26], authors investigated the size of the model and training time and noticed that by increasing the model size and training time, they can get better results.

After investigating previous transfer learning methods and trying different settings, the best approach is selected based on the results. The largest model that was achieved from combining the best methods had 11B parameters and achieved state of the art on CNN/Daily Mail, GLUE, SuperGLUE and SQuAD datasets. One interesting observation was that the model achieved close to human score on the SuperGLUE dataset which is a natural language understanding dataset.

T5 Modification on New NLP Tasks: T5 can be easily modified on other NLP tasks that are not mentioned in [26]. Here are two tasks that T5 is applied on:

1. **Question Answering**: Reading comprehension is a task where the model is given a context and a question about the context, and so the model should extract the answer of the question from the given context. T5 achieved state of the art results for this task on SQuAD dataset.

T5 was also trained on the close-book question answering task. In this task, the model is given a question without any context, and the model should find the answer to that question. Hence, the model should find the answer based on its pretrained knowledge that is stored in the model's parameter. T5 surprisingly achieved significant results on this task. T5 achieved the score of 50.1% on the Trivia-QA dataset, 37.4% on Web-Questions dataset and 34.5% on Natural-Questions dataset.The score of 50% means that 50% of the answers generated by the model, matches with the exact true answers to the question.

2. **Fill in the Blank Text Generation**: The pretraining objective for the T5 model is to predict missing words from the document. This made the authors in [26] to create a new task that should fill in the blank parts of the sentence with a specific number of words given to the model. For example in the sentence, "I like to play — in park." the model should fill in the blank with approximately words. By looking at the results generated by the model, we can conclude that the model has a perfect capability in the language modeling.

APPLICATIONS OF TRANSFORMER:

Transformers were first introduced for machine translation as a sequence to sequence model. Due to its strong performance and novel architecture, many researchers tried to utilize Transformers into various fields such as NLP, CV and audio processing.

1. **Natural Language Processing**: Transformers have been used in many NLP tasks such as machine translation [26–28], name entity recognition [29] and language modeling [30–32]. Pretrained transformers which are trained on large scale text corpora have pushed different NLP tasks into the state-of-the-art results.

2. **Computer Vision**: Transformers have been used in vision tasks such as image classification [33], video processing and object detection [34].

3. **Audio Programs**: Transformers have been adopted for audio related tasks like music generation [35], speech recognition [36, 37] and speech enhancement [38].

2.6 Tokenization

In NLP most of the data that we work on are raw texts. However, machine learning models cannot handle them as they just understand numbers. The process of converting raw texts into numbers is called tokenizing. Therefore, most NLP projects contain a tokenizer block at the beginning of their model to transform text (raw data) into numbers. There are several ways to perform this conversion and the goal is to find the most meaningful representation. We can categorize tokenizers into three main buckets:

- Word-Based Tokenizer
- Character-Based Tokenizer
- Subword-Based Tokenizer

2.6.1 Word-Based Tokenizer

Splitting a sentence into smaller pieces is not as easy as it looks. For instance, look at the sentence "Don't play soccer!". One simple approach to split this sentence into smaller chunks is to split up the sentence on each space we have in the sentence. As a result, we will get [Don't, play, soccer!] as the list of the tokens and then a specific number can be assigned to each token. At the first glance this looks good but the main problem is that punctuations are attached to the words, Like *"soccer!"*.

As a solution, We should teach the model to learn separate representations for punctuations as well. This will help to reduce the total number of words learnt by the model. Taking punctuation into account, the list of the tokens for the mentioned sentence will be [Don, ', t, play, soccer, !]. This looks better. However, the way

tokenizer dealt with the word *"Don't"* is the drawback for this method. *"Don't"* can be tokenized in different ways, ([Don,'t],[Do,n't],[Do, not]), depending on the rule we define for tokenization. This makes everything complicated because each model gets its best performance when the tokenization method used for its input is the same as the method used in its pretraining process.

Another problem that arises from using word-base tokenizers is that the two words *"dog"* and *"dogs"* get different numbers and therefore the model learns different embeddings for them. This is unfortunate because these two words have similar meanings and one of them is the plural form of the other.

In the word-based tokenizer, we should limit the total number of the words that the model wants to learn their embeddings in order to avoid extremely heavy models. As a result, we just take the most frequent words into the model and take the less frequent ones as out of vocabulary words and show them with the symbol *"UNK"*. Unfortunately, this idea introduces a new problem. For example it is possible that in a document, two words *"wanna"* and *"malfunction"* are not repeated frequently and so the model takes both as an *"UNK"* symbol in the vocabulary. However, these two words have entirely different meanings.

2.6.2 Character-Based Tokenizer

Character-based tokenizer is another model that can be used for tokenizing raw text. This model is very simple and decreases the model size and its time complexity significantly. In this model we split the text on each character instead of spaces and therefore individual characters get different ids. This idea has an advantage compared to the word-base tokenizer and that is smaller vocabulary size. The maximum vocabulary size that we get is 256 which includes all characters, numbers and special characters. Another advantage that we will get is less out of vocabulary words.

However, this algorithm has two significant drawbacks. The first one is that characters do not hold that much information individually as words will hold. For example,

learning an embedding vector for the word "hello" is much easier than finding a meaningful embedding vector for the character *"h"*.

The second problem of character based models is the length of its input sequences. As this model splits the input based on characters, the length of input sequences can be very large because sequences are translated into a significant amount of tokens. This reduces the size of the text that is fed to the model as input which is not good.

2.6.3 Subword-Based Tokenizer

To have a better understanding why subword tokenizer was proposed, we should take a look at the disadvantages of word-based and character-based tokenizers. In the word-based tokenizers, we end-up having a large vocabulary with lots of out of vocabulary tokens. In addition, this algorithm cannot distinguish between two tokens "apple" and "apples" and assign two different IDs to them. Therefore, this method learns two different representations for these similar tokens.

Character-based tokenizers end up producing long sequences as an input to the model which limits the length of the input sequence to the model. Another downside of this algorithm is that individual tokens are less meaningful because the model cannot learn too much about a token, based on its characters. To tackle these problems, subword tokenizer was proposed. The structure of this model is based on the followings:

1. Frequently used words should not be split into subword tokens

2. rare words should be decomposed into meaningful subwords

For example, for the word *"dog"* the tokenizer assigns a single id to it and doesn't split it. However, for the word *"dogs"*, the tokenizer decomposed it into two different tokens *"dog"* and *"s"*. Another example for this is the word *"civilization"* which is divided into *"civil"* as the root of the word and labeled as the start of the word, and

"ization" which is an additional information that changes the meaning of the root token and is labeled as the completion of the word.

Most models that achieve state of the art in the English language, use some type of subword tokenization. There are different models that can be used for subword tokenizers such as WordPiece (used by BERT and Distil-BERT [39]), Unigram (used by XLNet [29] and Albert [40]) and Byte-Pair Encoding (used by GPT-2 and RoBERTa).

The most important advantage of Subword tokenizer is reducing the vocabulary size while being able to learn meaningful context independent representation for different tokens. Now, let's talk about different kind of subword tokenizers

Byte-Pair Encoding (BPE)

This algorithm was first introduced in [41]. The first step in BPE is based on a pre tokenizer block that decomposes the training data into words. Different models use different pre tokenizer blocks. For instance, GPT-2 and RoBERTa use space tokenizer. XLM [42] and FlawBERT [43] use rule based methods and GPT which uses Spacy [44] to split training data into unique words and count them.

When pre tokenization is over, we should have a set of unique words with the number of times that they are repeated in the training data. Now, BPE creates a base vocabulary that contains all symbols that exist in the unique words. After this, BPE tries to learn merge rules to merge two symbols from the base vocabulary into one symbol and continue to do so until the vocabulary size is something desirable. It should be noted that vocabulary size is a hyperparameter that can be tuned during the training.

For the purpose of demonstration, Below is explained the workflow of BPE procedure [1]:

1. After pre tokenization, we have the set of following unique words with the mentioned frequency:

[1] https://huggingface.co/docs/transformers/tokenizer_summary

Table 2.2: Frequency of unique tokens

Word	**Frequency**
hug	10
kun	12
kug	5
bun	4
hugs	5

2. based on the unique words in the last step, we can build up the base vocabulary which is as follows:

$$Base\_vocabulary = [h, u, g, k, n, b, s]$$

As we have the base vocabulary built, the unique words in the first step can be split up into the symbols in the base vocabulary. This will give us the following:

Table 2.3: Frequency of unique symbols

Symbol	**Frequency**
h	15
u	36
g	20
k	17
n	16
b	4
s	5

3. In this step, BPE tries to find the combination of symbols that occur more often in the training data and make a new symbol out of them by joining them together. For example, *"ku"* occurs 17 times in the document (5 times in *"kun"* and 12 times in *"kug"*). However, the most common combination in the

document is *"ug"* which happens 20 times (10 times from *"hug"*, 5 times from *"kug"* and 5 times from *"hugs"*). Thus, *"ug"* will be the first symbol that is added to our vocabulary.

4. In this step, BPE goes over step 2 again and repeats steps 2 and 3 until the algorithm stops joining symbols. For instance, after adding "ug" to the vocabulary, our updated vocabulary looks like table 2.4:

Table 2.4: Frequency of unique symbols after one joining step

Symbol	**Frequency**
h	15
u	36
g	20
k	17
n	16
b	4
s	5
ug	20

As we can see from the table 2.4, *"ug"* is added to the vocabulary. BPE next tries to find another most common pair in the document. Here it is *"un"* which is repeated 16 times (12 times from *"kun"* and 4 times from *"bun"*). Therefore, *"un"* is added to the vocabulary. The next most common pair will be *"hug"* which is made from merging two symbols *"h"* and *"ug"*.Let's assume that the training of BPE ends at this point. Hence, we will end up with the following vocabulary:

$$updated\_vocabulary = [h, u, g, k, n, b, s, ug, un, hug]$$

5. As our vocabulary is built based on the merge rules during the training, BPE can be applied to new words. For example, the word "sun" can be represented

as [“s”, “un”]. Also, the word “pun” is tokenized to [“unk”, “un”]. As you can see, since the letter p is not in our vocabulary, we show it as an $< unk >$ symbol.

The size of the vocabulary is a hyperparameter and can be different for various models. For example, GPT has 478 base symbols and BPE finishes training after 40000 merges. So, it’s vocabulary size is 40478.

Byte-level BPE

In Byte-pair Encoding, the base vocabulary can become very large, especially if we consider special characters and unicodes in our vocabulary. To address this problem, GPT-2 utilizes byte as the base vocabulary. This would limit the size of the base vocabulary to be 256 and at the same time it includes all base characters in the base vocabulary. GPT-2 uses Byte-level BPE as its tokenizer and it can tokenize any sentence without requiring to add the $< unk >$ symbol to the vocabulary. After the merging process, GPT-2’s vocabulary size equals to 50257 which decomposes into 50000 for the merging numbers, 256 for the number of the base characters and 1 for the end of sentence token $< EOS >$.

WordPiece

WordPiece is tokenization method which is used in BERT and Distil-BERT and was first introduced in Japanese and Korean Voice Search [45]. This tokenizer is very similar to BPE with some minor differences. Just similar to BPE, WordPiece first builds a base vocabulary by including every character that exists in the training data. Next, it learns merging rules to merge the base characters together and add them to the vocabulary. However, the merging rule that is used here, is different from the one that BPE uses. As it was mentioned before, BPE chooses the merging rules based on the frequent symbol pairs, but WordPiece chooses the one that maximizes the likelihood of the training data.

But, what's the definition of maximizing the likelihood of the training data? Wordpiece, selects a symbol pair, which its probability divided by the probability of the first symbol followed by the second one, is the greatest among other symbol pairs possibilities. For example, in the case that was explained in the BPE section, *"u"* and *"g"* are merged just when the probability of *"ug"* divided by the probability *"u"* followed by *"g"* is higher than other possible symbol pairs.

Unigram

Unigram is another tokenization method which is defined under the subword-based tokenization category. This algorithm was first proposed in [46], Unlike BPE and WordPiece which the base vocabulary was initialized first and then new symbols were added by merging the symbol pairs, In Unigram, the base vocabulary is set to a very large size initially and then its symbols are removed gradually to obtain a smaller vocabulary. For example, the initial base vocabulary in Unigram can include all possible characters and pre tokenized words. It should be noted that Unigram doesn't remove the base characters from its base vocabulary, so that all new words can be tokenized.

But how does Unigram select the symbols that should be deleted from the initial base vocabulary? Unigram defines a loss over the training data considering the current vocabulary at each training step. Next, the algorithm computes the amount of loss increase for each symbol, if they would have been removed from the vocabulary set and then rank them. After calculating loss value for each symbol and ranking them, Unigram deletes p (which is around 10% to 20%) percent of the symbols which removal is the least compared to others.

As Unigram doesn't use merge rules for creating its vocabulary, each new word can be tokenized in different ways. Let's say the following vocabulary set is created after the training:

$$Vocabulary\_set = [s, u, n, a, b, c, d, sun]$$

Now we want to tokenize the word "sun" according to the vocabulary set mentioned above. We can do this in two different ways: ["s", "u", "n"], ["sun","s"]. But which one does unigram use? To answer this question, we should know that Unigram stores the probability of each token in the training data, besides storing the vocabulary set. By having each token's probability, Unigram can calculate the probability of each tokenization option and choose the most probable one.

SentencePiece

The next tokenization algorithm that we are going to cover is SentencePiece. SentencePiece tokenizer is a solution to tokenizing languages which their words are not separated by spaces. So far, all the described methods assume that words are separated by spaces which is not the case for all languages. SentencePiece was first introduced in [46]. This algorithm considers spaces as a character as well and therefore takes the input text as a full stream. Afterward, it uses BPE or Unigram to build its vocabulary. Different transformer models utilizes SentencePiece as their tokenization method which we can refer to T5 and ALBERT as an example of them.

2.7 Knowledge Graphs

Knowledge graphs (KGs) consist of some nodes that are connected together through some edges [47]. Both edges and nodes are labeled and the direction of the edges are from one node to another. Nodes can represent anything like people, organizations, places or cities. Each edge connects two nodes together via a relation that represents the semantic connection that exists between the nodes. For example, if $< Rome >$ and $< Italy >$ are two different nodes in a KG, then the relationship between these two nodes is $< Is\_the\_capital\_of >$ and its direction is from *Rome* to *Italy*. A pair of nodes that are connected together through an edge, are called a triple. In a general way, if we have a set of nodes E and a set of edges R, then a KG is a subset of E$\times R \times E$.

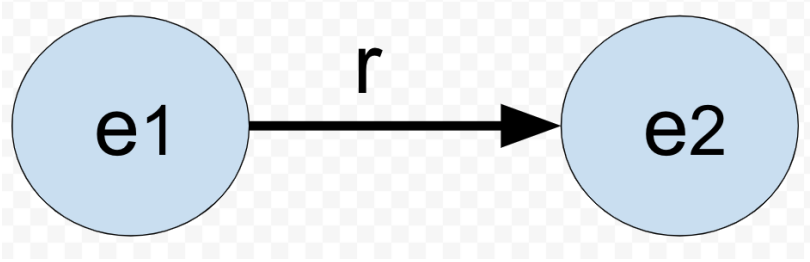

Figure 2.11: Triple representation as a graph

In the Figure 2.11, a triple is shown in a KG. $e1$ and $e2$ can be named differently based on the application of the KG. However, most of the KGs name $e1$ as a subject, $e2$ as an object and the r as a predicate.

When we move along different nodes in a KG, we are basically making a path. In a KG, different paths can be defined such as simple path and cycle path. Simple path is a path that is made when we move along different nodes but we don't cross repetitive nodes. However, in a cycle path, the first and the last nodes of the path are identical.

2.7.1 Resource Description Framework

Resource Description Framework (RDF) [48] is an essential part for data representation in semantic web, which was defined and developed by World Wide Web Consortium (W3C). In RDF, triples are used to represent and link different resources of data together. Each triple represents an RDF statement and can be in the form of $< subject, predicate, object >$ or $< entity, attribute, value >$. Subject and object in a RDF statement belong to the category of *(rdf:resources)* and are represented by a Uniform Resource Identifier *(URI)*. However, the predicate goes under the *(rdf:property)* category and is also represented using a URI. The predicate links the object and subject together using a relationship. It should be noted that sometimes the object can be a literal value and so does not have a URI.

One of the main problems of RDF is that subject, object or predicate can get anything that is asserted to them. This may cause creating RDF statements that

don't make any sense. For example $< Italy, has\_the\_nationality\_of, Rome >$. This triple can exist in RDF but doesn't have any meaning and is a false statement. To overcome this challenge, W3C introduced RDF Schema *(RDFS)* which defines some vocabulary on top of the RDF by creating classes and therefore restricts the URIs that can be asserted to a RDF statement.

2.7.2 Knowledge Bases

There are some Knowledge bases (KBs) available on the web which can be used as a source of knowledge for different applications such as question answering. These KBs put vocabularies and information in a structured way which makes it easy to search through their databse in the web. In the following, we introduce some of the famous KBs.

DBpedia

DBpedia [49] is manually created using the cross domain data taken from the structured parts of Wikipedia information such as info boxes and tables. It has a limited ontology and allows users to customize annotations to their individual needs.

YAGO

YAGO [50] is another KB that takes the knowledge from Wikipedia and combines them with the information that exists in the WordNet [51] and [52]. There are about 10 millions entities in YAGO and around 120 millions triples linking entities together.

Wikidata

Wikidata [53] is a big, controlled open domain knowledge graph that relies on crowdsourcing to populate data into the KB. The information that exists in the Wikipedia is also included into the Wikidata. The possibility for the people to change and edit the data easily is one of Wikidata's primary features. Another good thing about

Wikidata is that valid facts and triples from other sources can be integrated into its KB.

Freebase

Freebase [54] was a large KB that was controlled and edited by the community. There were many facts and information included in this KB. It had a diverse and cross-domain knowledge and at the same time, the size of the knowledge was huge. It had more than 2 billions triples which were related to a huge number of domains. However, this project stopped in 2016 but its data is available online.

2.8 Extracting Information from Text

Natural Language Processing (NLP) points to the processing of textual data and analyzing them. In order to inspect textual data, many NLP tasks have been introduced including question answering [55], machine translation [56] and information extraction [57].

Information Extraction (IE) is one of the important tasks in NLP that aims to extract meaningful information from plain text. IE converts unstructured text into a structured one by identifying existing entities in the text and their linking relations [58].

Structured information extracted by IE can be used to populate knowledge bases (KBs) such as DBpedia, Freebase and Wikidata. These KBs are a collection of different entities that are linked together through different relations. This structured format can be used in many NLP tasks such as question answering, relation extraction and search engines. Therefore, the final goal of IE is to take plain text as input and then extract relevant facts and information from the text and then upgrade or create knowledge based on these information.

2.9 Name Entity Recognition

Name Entity Recognition(NER) [59] is a crucial task in NLP that can be useful in many other applications including IE, question answering, information retrieval and text summarization. The purpose of NER is to identify and extract all possible entities from the text. Entities can be domain independent such as person, location and organization or they can be domain dependent such as medical entities including the name of drugs or the name of diseases [60].

Performing NER task may be challenging due to some reasons such as mixed entities, unclear text, missing data and different structures for different languages [61]. NER can be done in different ways such as Rule-Based methods, Learning-Based methods or a combination of both. However, deep learning methods such as CNN [62] are proven to have a better performance [63] in extracting entities.

Different works have been proposed to address NER task. [64] introduce a method to enhance the performance of NER for medical data. The dataset that is used contains 19378 samples of patient records. They suggest a model based on the combination of CNN, LSTM and CRF [65] to achieve better results. [63] proposed an algorithm based on the combination of CNN, Bi-LSTM and CRF to improve the scores of NER task. The main goal here was to perform NER without doing Featuring engineering. Their dataset was collected from online medical diagnosis records and contains 320000 samples.

[66] utilizes rule based methods such as grammar rules to Identify the entities in the text. However this approach achieves low precision. [67] suggests utilizing unsupervised machine learning methods such as clustering to extract entities. They examined their method on Facebook posts and noticed that spectral clustering achieves better results compared to other clustering techniques. In [68] some rules are learned to identify entities. This method can be called a hybrid method since the rules are generated automatically. The dataset used here is related to the medical domain

and is collected by Electronic Health records (EHR).[61] proposes a sequence labeling technique followed by a Support vector Machine (SVM)[69] classifier. The results demonstrate that CRFs achieve higher precision, however SVM obtains higher recall. In addition, CRFs are proven to have better performance in case feature engineering is not done well.

2.10 Relation Extraction

Relation Extraction (RE) is an important task in NLP and is a crucial part of knowledge graph construction. The goal in RE is to identify the set of relations linking different entities together in the text. However, extracting these predicates is not as easy as it shows, because textual data can be expressed in different formats. The performance of a relation extraction model, is highly dependant on its ability to understand the text.

Traditional approaches rely on feature engineering to extract the relations in text. The features are extracted based on words and expressions in the text. However this method requires lots of effort to define efficient features for each relation and also it may not perform well when the text is ambiguous.

Nowadays, deep neural networks have been used in many applications and they have shown prominent results in many tasks. Neural networks have non-linear activation functions which give them the ability to learn feature representation from the input data without the need to define the features manually. As a result, neural networks can learn different features, even the complex ones by their own nature. This concept has been used broadly for many NLP tasks including relation extraction.

RE can be classified into three different levels:

1. **Sentence-level RE** which identifies the relation that connects two different entities in a sentence.

2. **Document-level RE** which sometimes is called *"cross sentence RE"*. In this

category, the goal is to detect the relation between an entity pair that can repeated multiple times in a document and in different sentences.

3. **Corpus-level RE** which detects the relation between an entity pair without considering the text that includes the entity pair.

Some researches utilize other kind of method to conduct RE task, such as dividing the task into *Pre_defined Relation Extraction* and *Open Relation Extraction* which are explained below:

1. **Pre_defined RE**: In this type of RE, the set of relations to be extracted between pairs of entities, are known and the task is to pick one of them for each entity pair. This can be interpreted as a classification task.

2. **Open RE**: In this type of RE, there is no predefined set for the relations. Therefore, the extracted relations are textual expressions extracted from the text that can be interpreted as a relation.

2.10.1 Incorporating Neural Network for Relation Extraction

Neural Networks has shown to achieve strong results in. many NLP tasks. Different neural networks have been used to perform RE task in different researches. Convolutional Neural Networks, Long Short Term Memory Networks, Graph Neural Networks and attention based methods are different variation of neural networks. In the following we describe different researches that have utilized different types of neural networks for RE.

Convolutional Neural Networks in RE

A CNN based model has been proposed for RE in [70]. In this method, each token in the sentence has been converted into a low dimensional vector which is called word embedding. Different types of word embeddings can be used to represent each token,

such as: Word2Vec [71], Glove Embedding [16] or Facebook Fast-text embedding [72]. In addition to word embedding, relative positional embedding for each token is used and concatenated to word embedding. When vector representation is calculated for each token in the sentence, convolution operations are applied on token vectors to extract local features from the sentence. After extracting local features, we perform a pooling function to extract the most beneficial feature from each feature set.

Authors in [73] proposed a piecewise convolutional neural network (PCNN) to get better results compared to the method suggested in [70]. PCNN divides each sentence into three sections: before the first entity, between two entities and after the second entity. After this division, we apply a convolution filter and max pooling function separately on each division to extract the most important feature from each section. Another variation of CNN is proposed in [74] to extract sentence level features for RE. The model is called ranked based CNN (CR-CNN). They proposed a new ranking loss function which is useful to diminish the effect of fake classes and is described as follows:

$$L = log(1 + exp(\gamma(m^{+} - s_\theta(x)_{y_+}))) + log(1 + exp(\gamma(m^{-} - s_\theta(x)_{c_-}))) \tag{2.27}$$

In [75] a combined CNN with an attention mechanism is proposed to perform RE. The main goal of the attention mechanism is to extract the most important features from the sentence that is beneficial for RE. The attention structure is composed of two different levels. The first level can be described as follows:

$$\alpha_i^j = \frac{exp(A_{i,i}^j)}{\sum_{p=1}^{n} exp(A_{p,p}^j)} \tag{2.28}$$

$$r_i = z_i \frac{\alpha_i^1 + \alpha_i^2}{2} \tag{2.29}$$

The second level of attention mechanism has the purpose of pulling out more intellectual features from the sentence.

The previous methods do not incorporate syntax information into their model. For addressing this issue, in [76] a CNN model is proposed which integrates syntax

information as well. This helps the model to obtain a better interpretation of entities and boost the RE performance.

Recurrent Neural Network In RE

Just like CNNs, Recurrent Neural Networks (RNNs) can also be used to extract feature representation of a sentence and then utilize this feature representation to extract the valid relation in the sentence. In recent years, Long short term memory networks (LSTM) which are a variation of RNNs, have become popular models in various NLP tasks. LSTMs alleviate some of RNNs challenges including gradient vanishing problem by integrating gate function and cell memory into the model.

Standard LSTM analyzes sequences just in one direction (forward or backward). Therefore, Bidirectional LSTM (BILSTM) networks are proposed which can analyze sequences in both directions at the same time. Hence, they will produce a better feature representation from the input sequence and RE task can be executed better.

authors in [77] applied BILSTM on the input sentence to get the feature representation of both forward and backward directions. Then, these two representations are concatenated to each other to obtain the final feature representation of the sentence. The final representation is fed into a softmax classifier to extract the relation. In [12] a syntactic tree is built based on the input sentence using Stanford parser [78]. They applied LSTM on the built tree to obtain the feature representation of the input sentence. In addition to the syntactic tree, some other information is fed into the model, including part of speech tagging and WordNet hypernym. In [79] a new variation of LSTM is introduced which is called context aware LSTM. This model can learn the representation for all the relations that occur in a single sentence all in once. In [80] some LSTMs are combined together using adaptive boosting method. By applying this ensemble technique over LSTMs, the final model can learn semantic representation of the input sentence better and therefore we will get higher quality for relation extraction.

Attention Based Methods in RE

Attention algorithm is a very important concept in NLP that has been used in many tasks including machine translations and text classification. Therefore, some researchers have integrated attention into RE task. In [81], it is proposed that for extracting the target relation in a sentence connecting a pair of entities, Some words in the sentence are more important than the others and the model should put more weights on them while extracting the relation. In [82] BILSTM based attention mechanism model is introduced. In this model, first feature representation of the input sequence is extracted using BILSTM, and then an attention mechanism is applied on the features, so that the model puts more weight on the important sections of the sentence. In [83], authors proposed a new attention mechanism which is based on a hierarchy. The output of this hierarchical attention model is a 2 dimensional matrix which each of its row, focuses on a specific aspect of the input sequence.

Application of Transformers in Relation Extraction

Lately Transformers have become a very trendy method for various NLP tasks. Transformers were first introduced in [17] for machine translation and after that, it was used in other NLP tasks, which showed exceptional results. Most of the methods suggested for RE use static word embeddings such as GloVe or Word2Vec for their token representation. In recent years, contextualized word embeddings have gained interest among researchers. Contextualized word embeddings can be taken from different models such as BERT or ELMO. Utilizing these word embeddings and integrating them into the model, can boost the results for RE significantly.

In [84] a generative pretrained transformer (GPT) was employed to extract relations from the sentence. Transformers can learn semantic and contextualized representation of the sentence at a very high level, which helps the model to extract difficult and long relations as well.

In [85] BERT model is used to extract all the relations in the sentence along with

their corresponding entity pairs. In [86], BERT embeddings were used instead of GloVe for RE, and the results proved that contextualized embeddings increase the performance substantially. SpanBERT was proposed in [87] for the RE task. They used BERT architecture as the backbone of their model with some modification. For detecting the valid relation between head and tail entities in the sentence, a linear classification model is added on top of the *[CLS]* token. In [88], a fine tuned BERT model is used for RE task. In the first part, they fine tuned BERT to see if there is any relation in the sentence connecting entity pairs together. If in the first step the model confirms a relation, then in the second part BERT is fine tuned again to identify the relation.Authors in [89], utilized BERT for document level RE and the results were outstanding.

Graph Based Neural Networks in RE

LSTMs and CNNs are good fit for sequential data. However, some data doesn't have sequential nature such as the dependency tree which is obtained from the sentence. Dependency tree has a graph-like representation and it contains rich syntactic information about the sentence. To analyze this type of data, we need Graph Neural Networks(GNNs) [90].

In [91] Graph Convolutional Network (GCN) is employed for relation extraction. They first build the dependency tree of the input sentence and then pool information from the dependency graph in a parallel manner. In [92] the attention mechanism is combined with GNN to choose the most relevant parts of the dependency tree for relation extraction. In [93] another variant of GNNs is introduced which is called edge oriented graph neural network. This model builds a graph based on various types of edges. In [94] relation extraction is applied on the medical domain. For this purpose, they build a dependency forest to extract input features. To analyze the dependency forest, GNN is utilized to select the relevant parts of the forest and remove irrelevant parts.

Incorporating Few Shot Learning in Relation Extraction

Some researchers have recommended using a few shot learning model for relation extraction. In [95] few shot learning is incorporated into RE. They proposed a prototypical network in their model and also applied attention in sentence level as well as relation level to choose those sentences that are actually representing a relation. In [96] few shot learning is used to learn the relation features from the plain text. In [97] few shot learning is utilized to learn new relations from few gold training sentences representing those relations. They utilized the bootstrapping method to transfer the knowledge from already learnt relations into the process of learning the new relations. By doing so, the model can learn the new relations by just looking at a few sentences and therefore find all relevant sentences from the unlabelled dataset.

2.10.2 Weakly Supervised Methods for Relation Extraction

In this section, we will describe weakly supervised methods and investigate different solutions to handle weakly supervised dataset. In these methods, training data are generated using some heuristic rules and some predefined triples in the knowledge base to label sentences with different relations. But as we use heuristic rules to label the sentences, we will have noises in our labeling process. To address these noises, different approaches have been presented which we will go over in this section.

Distant supervision approach is proposed in [98] to label the training instances. In this method, they assume if $<e1,\ r,\ e2>$ is a triple in the knowledge base, then all the sentences that contain *e1* and *e2*, express the relation r and so can be labeled as r. All the sentences that contain *e1* and *e2* will go in one set and the set is called a bag. However, distant supervision introduces noise into the training data. The main problem of this method is that, not all the sentences that contain *e1* and *e2*, necessarily express the relation r. But, distant supervision labels all these sentences with the relation r. To alleviate the effect of noisy sentences, different approaches have been proposed.

Multi Instance Learning

Multi Instance Learning assumes that at least 1 sentence in the bag represents the true relation. In [73] Multi Instance Learning is used in their proposed model. They utilized PCNN to extract feature representation of the sentence and then choose a sentence from the bag which most likely represents the valid relation as the training data and ignore the remaining sentences. This can be formulated as follows:

$$J(\theta) = \sum_{i=1}^{T} logp(y_i|m_i^{j^*};\theta) \tag{2.30}$$

$$j^* = argmax_j p(y_i|m_i^j;\theta) \tag{2.31}$$

Where m_i^j represents the j_{th} sentence in the bag i and T is the number of the bags.

Attention Mechanism

Selecting just one sentence out of the bag for training, leads to a significant amount of information loss which may ruin the model's performance. For addressing this issue, sentence level attention mechanism is proposed in [99, 100]. They showed that by applying attention over the bag sentences, noisy sentences are assigned with smaller weight but on the other hand, true sentences that represent the true relation get higher weight. In [101], word and relation level memory networks are proposed to increase the accuracy of attention's weight computation.

Reinforcement Learning

Some papers included Reinforcement Learning (RL) [102] into RE task. Instead of relying on attention weights to remove noisy instances from training data, RL methods use a hard selection strategy. In [103] A RL method is proposed that includes two different parts: sentence picker and relation selector. The sentence picker part, picks true labeled sentences from a bag and passes them to the relation selector part.

Relation selector part selects the valid relation for these sentences and gets a reward. This reward is calculated based on the probability scores for each relation.

[104] introduced a deep RL method for relation extraction. The main difference of this model with the one presented in [103], is the reward calculation process. In [104] the reward of RL is calculated based on how well the relation selector part is doing. More its performance improves, the higher the reward for the model. [105] proposed a method to take false negative instances into consideration. They believed that some of the false negative instances are labeled wrongly and they actually have a valid relation to classify. In their model, noisy data are divided into correctly labeled data and wrongly labeled data. The model learns to separate them via reinforcement learning and so improves the quality of the labeled training set. In [106] RL is used to determine the relation of a bag. The process is in a way that if all the sentences in the bag represent the relation *NA*, then the bag also takes the *NA* relation. However, if just one of the sentences in the bag doesn't represent the *NA*, then the bag is transferring some semantic information and can be labeled as a valid relation. When the model assigns a relation to a bag, the reward of the RL model is calculated by comparing this relation to the gold relation of the bag.

2.11 Triple Extraction

Triple extraction means extracting a pair of entities (head and tail) and the corresponding relation between them from a sentence or document. These triples can be used to enrich different KBs such as DBpedia or Wikidata. KBs contain many different triples that look like a database of information. In most of the existing KBs, triples are added to the database using crowd-sourcing. Therefore, adding new triples to the database or updating it, requires so much time and hard-work. Therefore, researchers were looking for a way to extract triples from the text automatically without human intervention. This will make it much easier to add new triples to KBs or build a KB from a scratch.

2.11.1 Task Description

Triple extraction can be described as a box that takes a sentence and a set of relations *"R"* as input, and the output is a set of triples *"T"* extracted from the sentence. Some methods are pipeline based which means they divided the task into two different assignments:

1. Identifying all entities in the text which can be called named entity recognition task.

2. identifying the valid relations between different entities which can be named as relation classification task. It should be noted that between some entities there is no relation and so in this case *"NA"* relation is assigned.

Beside pipeline approaches, there are joint-extraction approaches as well which extract entities and relations in a joint manner. Joint methods extract all possible triples from the text all in once. In these methods, *"NA"* relations won't be extracted and hence, just valid triples are identified. Extracted triples can be divided into three different classes:

1. **No Entity Overlap (NEO)**: In this case one or more triples are extracted from the text but their entities are totally different from each other.

2. **Single Entity Overlap (SEO)**: In this case more than one triple is extracted from the text, however two or more triples share one entity among each other.

3. **Entity Pair Overlap (EPO)**: In this case more than one triple is extracted from the text, however two or more of them share the same entity pair. The shared entity pair can be in the same order or reverse.

These three classes are shown in the Table 2.5.

Table 2.5: Different triple categories based on entity overlap

Class	**Text**	**Valid Triples**
NEO	Sahand was raised in Sanandaj.	$(Sahand, place\_lived, Sanandaj)$
SEO	Alex was born in Tehran but raised in Paris.	$(Alex, place\_of\_birth, Tehran)$, $(Alex, place\_lived, Paris)$
EPO	Paris is the capital of France.	$(France, capital, Paris)$, $(France, contain, Paris)$

2.11.2 Triple Extraction Methods

In the previous sections, we talked about relation extraction (RE) methods and we mentioned that RE approaches assume that entities are already assigned in the sentence and so, they just do the relation classification part between pairs of entities. However, the accuracy of the extracted triples in this case, highly depends on the name entity recognition system. To overcome this issue, joint extraction approaches are proposed which extract entities and relations at the same time. [107, 108] used the same network to extract entities and relations. In the first step, they extracted entities from the text and in the second step identified a relation between all pairs of entities. In this case the name entity recognition model and RE model share the same parameters. However, these methods still look like pipeline approaches and ignore the possible connection between the two steps.

In [109] a sequence tagging model is proposed to extract triples from the text. They create a set of tags that include entity and relation information. The main drawback of this method is its inability to extract triples that have entity overlap. An encoder-decoder method is proposed in [110], which is combined with a copy mechanism. Their model is able to extract triples including overlap triples, however, they need to use a separate decoder for each extracted triple. The main problem of this approach is setting a value for the number of decoders in the model. In addition,

the model cannot extract the full entity token and just extract the last part of the entity. For solving this issue, CopyMTL model was proposed in [111] which utilizes sequence tagging framework. This helps the model to extract the full entity and not just a part of it.

Graph convolutional network (GCN) is proposed for triple extraction in [112]. In this model a graph is built in a way that its nodes are sentence tokens and its connections are relation between different tokens. In [113], authors proposed an encoder decoder model that a N gram attention approach is integrated into the model. In [114] an encoder decoder network is proposed for triple extraction where the decoding is at the word level. They also utilized a pointer model in their method. In [115] the triple extraction task is divided into two different steps. The first step is extraction of the head entity in the triple, and the second step is extraction of the tail entity and the relation together. In [116] a sequence labeling framework is proposed for this task that integrates attention model into the relations. In [117], a reinforcement learning (RL) method is proposed. They applied RL on the input sentence in two steps. In the first step, RL is responsible to extract the relations from the sentence and in the second step, RL is responsible to extract the head and tail entities for the identified relation in the first step. Sequence labeling framework is applied to extract the entities. They will go over this procedure many times so that all the triples are identified in the sentence.

In [118], a new loss function is introduced so that the performance of triple extraction is improved. They called the function bipartite matching loss and applied it on a encoder decoder model. In [119] a model is proposed which is based on two different encoders, sequence encoder and table encoder. The former one is responsible for identifying the entities in the sentence and the latter one is in charge of relation extraction. The main problem of pipeline approaches for triple extraction was neglecting of the possible interconnection between entity recognition and relation classification. For solving this issue, [120] proposed a multi-task learning framework

that considers the possible connection of these two steps in the model. In [121], the input sentence is divided into different spans and in each span, a multi head attention mechanism is applied.

Some methods [118, 122–124] have employed BERT embeddings in their model for triple extraction. In these models, the vectors of the last layer in the BERT network are used for token representations instead of static word embeddings such as GloVe. By doing this, the results for triple extraction get boosted remarkably.

Chapter 3

Relation Extraction with Sentence Simplification Process and Entity Information

Graph-based Knowledge Bases (KBs) are composed of relational facts that can be perceived as two entities, called *head* and *tail*, linked via a relation. Processes of constructing KBs, i.e., populating them with such facts, as well as revising and updating them are of special interest. These should be performed automatically, especially in the case when the main sources of facts are textual documents. For this reason, a task of Relation Extraction (RE), i.e., predicting a relation that links two entities mentioned in a sentence, is one of the most important activities. Using RE processes, new relational facts can be extracted, and KBs can be built and updated using unstructured information. In this chapter, we propose a novel procedure for RE. It is based on a sentence distilling technique that works on dependency trees and removes noisy tokens from sentences while preserving the most relevant and useful ones. In addition, the proposed procedure utilizes information about types of linked entities, it means types of relations' *head*s and *tail*s. Our neural network model using processed and new input information is evaluated on the widely used NYT dataset and compared to other state-of-the-art RE methods. Experimental results show the effectiveness of the proposed procedure against other methods.

3.1 Introduction

Knowledge Bases (KBs), such as Freebase [2], DBpedia [3] or Wikidata [4], are well-known repositories of information triples [1]. Each triple is a pair of entities – *head* and *tail* – linked together via a relation. Semantically rich information stored in KBs is used in various application domains tasks such as biomedical knowledge discovery [7], semantic search, and question-answering systems [5]. However, the information stored in KBs is not complete and should be updated on regular basis. When new information becomes available, it should be added to KBs while old information should be revised or discarded. Therefore, processes of constructing triples – <*head, relation, tail*> – and adding them to KBs in a structured form are necessary. Relation extraction (RE) is an activity that addresses this by extracting relations between entity pairs from a plain text. RE can be modelled as a classification task that assigns a relation to a pair of entities mentioned in the text.

Most of the supervised RE methods require large amounts of annotated training data, which is very time consuming and not efficient to build. [98] proposes a distant supervision to automatically label training data based on existing information available in KBs. The method assumes that if <*head, relation, tail*> is a relational fact existing in a given KB then all sentences that include entities *head*s and *tail*, are linked with the relation *relation*. For example, <*Barack Obama, place-of-birth, Kenya*> is a triple. The distant supervision assumes that any sentence that contains *Barack Obama* and *Kenya*, expresses the relation *place-of-birth* and can be taken as a training data for the relation *place-of-birth*.

Although the distant supervision is a convincing method for labelling instances, it suffers from a problem of incorrect labelling. For instance, any sentence which contains *Barack Obama* and *Kenya* does not necessarily express the relation *place-of-birth*. For example, *"Barack Obama travelled to Kenya last year"* contains both entities, but we cannot infer that this sentence convey *place-of-birth* relation. Yet,

distant supervision recognizes this sentence as a part of a training dataset. To alleviate this problem, [125] adopt multi-instance learning. The main drawback of these methods is that they use NLP tools such as POS tagging for extracting features. These tools are not error free and their mistakes hinder the RE performance.

Auxiliary information, such like the type of entities, can improve the RE performance. For example, in the relational fact <*Barack Obama, place-of-birth, Kenya*>, *Barack Obama* is an entity of type *Person* and the type of *Kenya* is *City/Country*. Therefore, in any sentence that expresses the relation *place-of-birth*, the pair of entities should be of the type *(person, city/country)*. Any constrains imposed on types of entities provide an extra support in extracting the correct relation from a sentence.

Dependency parsers find syntactic dependencies between different tokens of a sentence. Rich structural information embedded in dependency trees, have been proven to be useful for RE tasks. Yet, the full information provided by a dependency tree is not useful and in some cases it may introduce unneeded noise. To address this, we can breakdown the tree into smaller parts and crop a substructure of the tree containing the most valuable information required for RE. By doing this, we pick the most useful tokens from the sentence and ignore the rest. For example, let us take a look at the sentence *"Three years ago when I was 13,* ***John*** *was born in* ***Canada****"*. For the purpose of extracting the triple <*John, place-of-birth, Canada*>, it is enough to use *"John was born in Canada"* instead of the full sentence.

In this chapter, we propose a neural network model to improve the distant supervision RE process. We develop a procedure using syntactic information obtained from dependency trees to remove noisy tokens from sentences and pick the most relevant ones that help extract correct relations. This method also utilizes entity type information, which is fed to the proposed model. Entity type information is obtained from KBs. We evaluate our model on the NYT benchmark dataset. The experimental results show that our new model leads to a significant improvement over the other state of the art neural models for the RE task. The summarized contribution of this

chapter is as follows:

- utilization of dependency trees to eliminate irrelevant and noisy tokens from sentences in order to keep only relevant words useful for the RE task;
- inclusion of entity type information, obtained from KGs, as an entity type embedding to the neural network model.

3.2 Related Work

Relation extraction is an important task in Natural Language Processing (NLP). Performing RE tasks in a supervised manner requires large amounts of labelled training data. To solve this issue, [98] propose distant supervision, which heuristically aligns plain text with relational facts from a KB like Freebase. This creates a large annotated dataset. However, this method suffers from mistakenly labelled sentences. [126] addresses this by using a multi-instance single label technique, while [125] propose a multi-instance multi-label procedure. All these approaches require feature engineering, and this can affect their performance negatively.

To avoid feature engineering, neural network models have been proposed. They have shown favourable results on RE tasks. [70, 73] extract sentence features using convolutional neural networks (CNN) instead of relying on traditional NLP tools. [70] propose an end-to-end CNN which can extract sentence-level features automatically. [73] introduce a piecewise CNN (PCNN) that breaks down a sentence into three parts: before the first entity, between two entities, and after the second entity; and extracts features for each part individually. The authors use a multi-instance learning strategy but assume that only one sentence holds the true relation for each entity pair. This would to the lost of information from neglected sentences. [127] use another popular architecture recurrent neural network (RNN) for the RE task. However, RNNs are affected by a vanishing gradient problem. [107] employ Bidirectional Long Short Term Memory (BiLSTM) to alleviate the RNNs problems. To alleviate introduction of noisy

data seen in distant supervision, attention mechanism is used. [82] use attention to learn about a given entity pair based on all valid sentences. These methods use full sentences, including irrelevant parts, to extract relations between pairs of entities. In addition, attention mechanisms are applied on sequences of words and are not able to capture information about their grammatical structure [128].

Dependency trees are able to capture syntactic information which is not local [91]. Employing this information obtained from dependency trees has shown prominent results. [129] utilize dependency tree in their deep learning model to jointly predict dependency and semantic relations. Some models use pruning strategies to extract the most relevant information from dependency trees. [107] propose Lowest Common Ancestor (LCA) between entities of a sentence to reduce a full tree to a subtree. Some models apply graph convolutional networks (GCNs) over a subtree to enhance the RE performance [91].

[100] use entity descriptions extracted from KBs to improve the RE results. [130] use entity type information in a model they propose. [131] employ KB embedding to link an RE task with a process of KB embedding.

3.3 Problem Statement

In this section, we provide a brief description of distant supervision which is used to annotate datasets. We also formally specify the problem we address in the chapter.

3.3.1 Distant Supervision for RE

A distant supervision process can be presented as a process of preparing data by labeling sentences with relational facts from a known Knowledge Base (KB). Let Φ be a KB composed of triples $(h(head), r(relation), t(tail))_\Phi$ where $h, t \in \mathcal{E}_\Phi$ and $r \in \mathcal{R}_\Phi$. Further, $\mathcal{E}_\Phi$, $\mathcal{R}_\Phi$ are sets of entities and relations defined in the Φ, respectively. If $(h, r, t)_\Phi$ is a relational fact in the Φ, then each sentence that contains both entities h and t, is labelled with the relation r. In order to mitigate a problem of incorrect

labelling, a set of sentences $S_r = \{s_i\}_{i=1}^{N_r}$ where each s_i mentions both h and t is called a bag, and the whole bag is labeled with the relation r. Bags can have different sizes, N_r, yet for the purpose of computational efficiency, we normalize each bag to the length of T by splitting larger bags and oversampling smaller ones [131].

3.3.2 Problem Definition

The goal of RE is to predict a relation, defined in the Φ, that links two entities in a sentence. Given a set of sentences $S_r = \{s_i\}_{i=1}^{N_r}$ and a pair of entities h and t mentioned in each sentence $s_i \in S_r$, the aim is to predict the probability $p\left(r|h,t,\{s_i\}_{i=1}^{N_r}\right)$ of assigning $r \in \mathcal{R}_\Phi^* \cup \{\mathcal{NA}\}$ to each bag, i.e., each set S_r. $\mathcal{R}_\Phi^*$ is a subset of relations $\mathcal{R}_\Phi$ defined in the KB Φ, while $\mathcal{NA}$ represents a relation that does not exist in R_Φ. The input to the RE task is composed of: a KB Φ, a subset of relations defined in the KB $\mathcal{R}_\Phi^*$, and a training dataset with annotated automatically sentences using distant supervision. As a result of the task, a relation $r \in \mathcal{R}_\Phi^*$ that holds for h and r is assigned to a bag, or the bag is labeled with $r = \mathcal{NA}$ if its sentences do not have a relation from $\mathcal{R}_\Phi^*$.

3.4 Methodology

This section includes an explanation of our proposed model. First, a new distilling strategy is introduced. Its goal is to extract the most useful tokens from sentences based on analysis of their dependency trees. Once sentences are simplified, vector representations of sentences and groups of sentences, i.e., bags are determined. Finally, we describe incorporating information about types of relation entities in order to improve the RE task. Figure 3.1, depicts the proposed model.

3.4.1 Sentence Simplification

Dependency parsers have been proven to be useful for the RE task. They provide an important information about agents and actions contained in sentences. Dependency

Algorithm 1 Sentence Simplification Algorithm

Input: sentence string S_i
head h and tail t
set of Core Arguments: Nominals and Clauses:
$CoreArg$ = { `nsubj`, `obj`, `iobj`, `csubj`, `ccomp`, `xcomp` }

Output: simplified sentence $SmpS_i$

generate dependency tree: $DP_i \leftarrow S_i$
$p \leftarrow path(h \text{ to } root \text{ in } DP_i)$
$set_H = string2set(p)$
for each $w_j \in set_H \setminus root$ **do**
    $set_H \leftarrow set_H$ + child of w_j
$p \leftarrow path(t \text{ to } root \text{ in } DP_i)$
$set_T = string2set(p)$
for each: $w_k \in set_T \setminus root$ **do**
    $set_T \leftarrow set_T$ + child of w_k
create set set_{DT} of words w_m at nodes from $CoreArg$
for each: $w_m \in set_{DT}$ **do**
    $set_{DT} \leftarrow set_{DT}$ + ancestor of w_m
$set_{All} = set_H \cup set_T \cup set_{DT}$
$SmpS_i = [\ ]$
for each w_p from S_i **do**
    if $w_p \in set_{All}$ **then**
        $SmpS_i \leftarrow SmpS_i.append(w_p)$

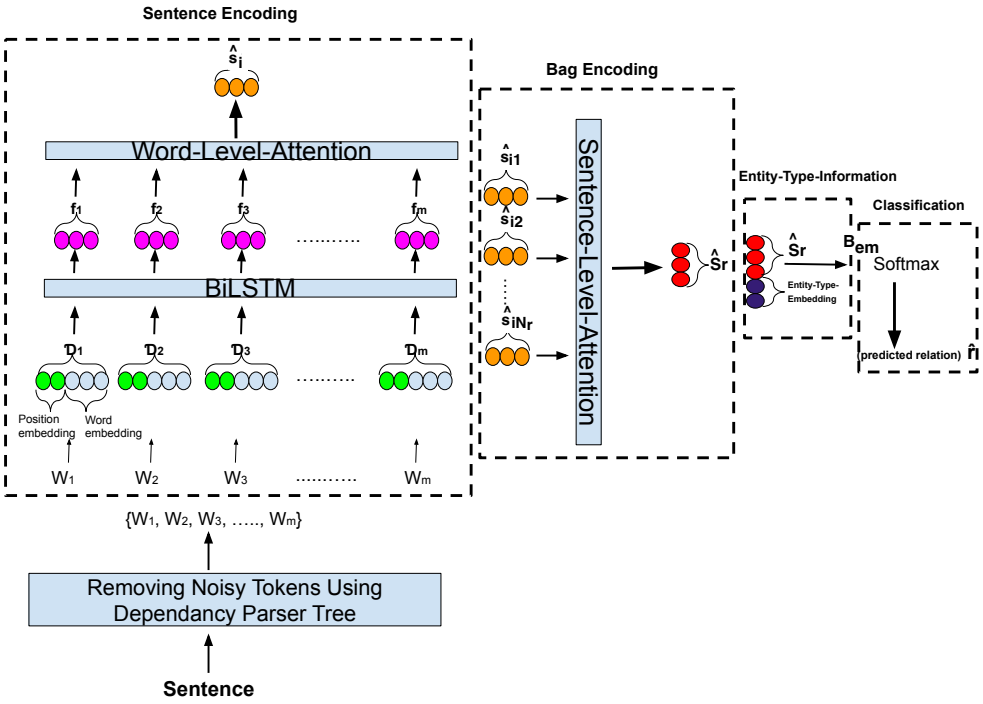

Figure 3.1: Outline of the proposed method. Each sentence is simplified using the proposed distilling strategy. Word embedding and position embedding of each token in the simplified sentence are combined together and fed into the BiLSTM. Sentence embedding for each sentence is obtained by applying word level attention over output vectors of the BiLSTM layers. After that, each bag of sentences is encoded via applying sentence level attention over the embeddings of all sentences of the bag. Finally, bag embedding is combined to type embedding of head and tail entities; all this is fed into a softmax classifier to predict a relation for the bag.

paths built by traversing the tree are informative. We propose and describe a new method to refine sentences via processing their dependency trees and extracting the most relevant tokens. This diminishes the impact of less relevant – noisy – tokens. To better explain our method, take a look at Figure 3.2. It contains a dependency tree of the sentence *"But now **Spanish** entrepreneurs want to join **Europe**'s boom in large wind farms offshore."* where *Spanish* and *Europe* are head h and tail t entities. Our sentence simplification method, shown as Algorithm 2, can be explained in the following way.

L: 2-6 Starting at the token representing *head_entity* h, we go up the tree to the root node, *want* in our case. The obtained path is ***(Spanish*** $\rightarrow$ ***entrepreneurs***

$\rightarrow$ ***want)***. We create $\mathbf{set_H}$ of all tokens from the path. For each word in $\mathbf{set_H}$, we add tokens of its children, except for the tree root (*want*). *Spanish* does not have any children tokens. For the word *entrepreneurs*, the child token is *Spanish* which is already in $\mathbf{set_H}$. As a result, we obtain: $\mathbf{set_H}$=**{*Spanish, entrepreneurs, want*}**.

L: 7-11 We repeat the previous step for the tail *entity_token t*. The obtained path is ***(Europe*** $\rightarrow$ ***boom*** $\rightarrow$ ***join*** $\rightarrow$ ***want)***. We create $\mathbf{set_T}$. Similar to the first step, for each token we add its children tokens. In the presented case, children tokens for *Europe*, *boom* and *join* are (*'s*), (*Europe*) and (*offshore, to, boom*) respectively. We have: $\mathbf{set_T}$=**{*Europe, 's, boom, join, offshore, to, want*}**.

L: 12-15 In this step, we analyze dependency relation types of sentence tokens and create a set $\mathbf{set_{DT}}$. If a dependency relation type of a given token is one of the core arguments of universal dependency relations, such as: *nsubj*, *obj*, *iobj*, *csubj*, *ccomp* or *xcomp* [132], we add this token to $\mathbf{set_{DT}}$. As it is shown in Figure 3.2, the dependency relations for the three words *entrepreneurs*, *join*, *boom* are *nsubj*, *xcomp* and *obj*, respectively. In addition, for each of these words, their ancestor tokens are also added to the set. The ancestor tokens for *entrepreneurs*, *join* and *boom* are *want*, *want*, and *join*. So: $\mathbf{set_{DT}}$=**{*entrepreneurs, join, boom, want*}**.

After performing all three steps, the final task is combine the sets: the final set is $\mathbf{set_{all}}=(\mathbf{set_H} \cup \mathbf{set_T} \cup \mathbf{set_{DT}})$={***Spanish, entrepreneurs, want, to, join, Europe, 's, boom, offshore***}. Finally, the distilled sentence which is fed to RE model is created based on $\mathbf{set_{all}}$ (Algorithm 2, lines 14 to 17). In our example it is *"**Spanish** entrepreneurs want to join **Europe**'s boom offshore"*. Predicting a relation between two entities *Spanish* and *Europe* in this simplified sentence is much easier due to removing irrelevant, noisy tokens from the original sentence. In this work, Stanza [133]

is used to construct a dependency tree for each sentence.

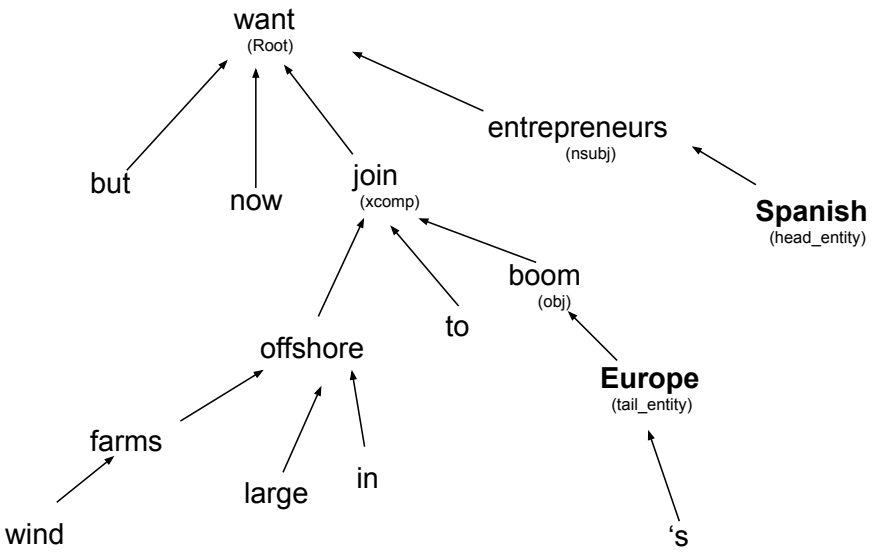

Figure 3.2: dependency tree for the sentence *"But now **Spanish** entrepreneurs want to join **Europe**'s boom in large wind farms offshore."*

3.4.2 Sentence Encoding

After simplification, we construct a vector representation for each sentence. For the sentence s_i with m tokens $\{w_1, w_2, ..., w_{\mathrm{m}}\}$, each token is represented with the pre-trained d_{w}-dimensional GloVe embedding [16]. Based on the relative position of each token in the sentence with respect to head h and tail t entities, each word's embedding is concatenated with $2 * d_{\mathrm{p}}$-dimensional position embedding [70]. By stacking both word and position embedding, we obtain the vector representation $D \in \mathbb{R}^{m * (d_{\mathrm{w}} + 2 * d_{\mathrm{p}})}$ of all tokens of the sentence. Afterward, D is fed to a bidirectional LSTM with a hidden unit size of d_{h}. As a result, we obtain an output for each layer of the LSTM $F = [f_1, f_2, ..., f_m]$, $F \in \mathbb{R}^{m * 2d_{\mathrm{h}}}$.

After obtaining the output vectors, word-level attention mechanism is applied over

$\{f_i\}_{i=1}^{m}$. For each vector $f_{\mathrm{i}} \in \mathbb{R}^{1*2d_{\mathrm{h}}}$, an attention weight is calculated as follows:

$$\alpha_{\mathrm{i}} = \frac{exp\,(u_{\mathrm{i}})}{\sum_{j=1}^{m} exp\,(u_{\mathrm{j}})} \quad \text{where,} \quad u_{\mathrm{i}} = f_{\mathrm{i}}.q \tag{3.1}$$

here, q is a query vector, u_{i} is the score assigned to each token, and α_{i} is attention weight calculated using softmax over u_{i} scores. The final representation of the sentence s_i after applying the word-level attention is obtained:

$$\hat{s}_i = \sum_{i=1}^{m} \alpha_{\mathrm{i}} * f_{\mathrm{i}} \tag{3.2}$$

with $\hat{s}_i \in \mathbb{R}^{1*2d_{\mathrm{h}}}$ and m the number of sentence tokens.

3.4.3 Bag Encoding

After encoding each sentence and obtaining its feature vector, a feature representation for each bag is established. As explained earlier, each relational fact in the KB Φ, which is in the form of triple $(h, r, t)_{\Phi}$, is used to label a set of sentences that all contain both entities h and t. These sets of sentences form a bag and the whole bag is labelled r. Each bag includes different sentences and some of them may be labelled wrongly as our dataset is distantly supervised. To alleviate wrong labelling problem, a sentence level attention mechanism is used to combine the embedding vectors of these sentences and drive the embedding representation for the entire bag. Attention weight for each sentence in the bag is calculated as below:

$$\alpha_i^{'} = \frac{exp\,(\hat{s}_{\mathrm{i}}.p)}{\sum_{j=1}^{N_r} exp\,(\hat{s}_{\mathrm{j}}.p)} \tag{3.3}$$

where $\hat{s}_{\mathrm{i}}$ is the feature vector for each sentence in the bag, p is a random query vector, and N_r is the number of sentences in the bag. The final representation for the bag is:

$$\hat{S}_r = \sum_{i=1}^{N_r} \alpha_i^{'} * \hat{s}_{\mathrm{i}} \tag{3.4}$$

with $\hat{S}_r \in \mathbb{R}^{1*2d_{\mathrm{h}}}$.

3.4.4 Entity Type Information

Each relation defines permissible types for its head and tail entities. For example, the relation ***Place-of-Birth*** defines a head entity to be of a type ***Person*** and a tail entity to be of a type ***Location***. As we use distant supervision for labelling a dataset from a particular KB, the information about types of relation entities can be obtained from the KB.

In this work, 14 different entity types taken from Stanza [133] are used. Each entity type, is converted into an E_d-dimensional embedding. The types for h and t could differ between sentences from a single bag. Therefore, we average type embeddings over all sentences from the bag. For example, *Canada* can be ***Location*** in one sentence and ***GPE(country)*** in another. So for this example, an average over the embeddings of types ***Location*** and ***GPE*** is used as final type embedding for the entity type of the word *Canada*.

The obtained type embeddings of head and tail entities for each bag are concatenated with the final bag representation $\hat{S}_r$ (see the previous section) and B_{em} is obtained:

$$B_{em} = [\hat{S}_r; \textit{head-type-embedding}; \ \textit{tail-type-embedding}]$$

B_{em} will be fed to the softmax classifier to predict the probability for each relation. The predicted relation is:

$$\hat{r} = argmax\ p(r|h, t, S_r; \Theta) \tag{3.5}$$

$$r \in R^*_\Phi \cup \{\mathcal{NA}\} \tag{3.6}$$

where S_r is the set of sentences for the bag linked with the relation r, and Θ are parameters of the model that are learned during the training process.

3.5 Experiments

3.5.1 Dataset

In this thesis, we utilize the **NYT** [126] dataset, which is widely used for the RE task. This dataset is annotated using the distant supervision method by aligning relational facts from Freebase to plain texts in New York Times articles. Articles from the years 2005 - 2006 are used for training, while articles from the year 2007 are used for testing.

3.5.2 Evaluation criteria

Similar to the previous work [98], the proposed model is evaluated using held-out evaluation. This means, relations predicted from our model are compared to the Freebase relations. *Precision* and *Recall* can be calculated without conducting time-consuming human evaluation. We measure the performance by plotting the *Precision-Recall* curve and reporting the *top-N Precision* (*P@N*) metric. It should be noted that the probability predicted for an $\mathcal{NA}$ relation is neglected while reporting the results.

3.5.3 Hyperparameter Settings.

We randomly select 15% of the training dataset and treat it as a validation set. This validation set is used for tuning hyperparameters of the model. With each specific hyperparameter, we run the model five times and report the average *P@N*. The model with the highest average will be selected to determine the values of hyperparameters. With the well-tuned hyperparameters, the model will be trained over all the training dataset.

3.5.4 Baselines Evaluated.

To evaluate the proposed model (**Proposed-Model**), we compare it against other methods including the previous state of the art neural models, i.e., **PCNN** [73],

PCNN+Attention [134] and **BiLSTM+Attention** [82]. We also evaluate the performance of a simpler version of the proposed model (**Proposed-Model-Simple**), which incorporates only the proposed distilling strategy, i.e., simplification of sentences by deleting noisy and irrelevant tokens, without incorporating the entity type information.

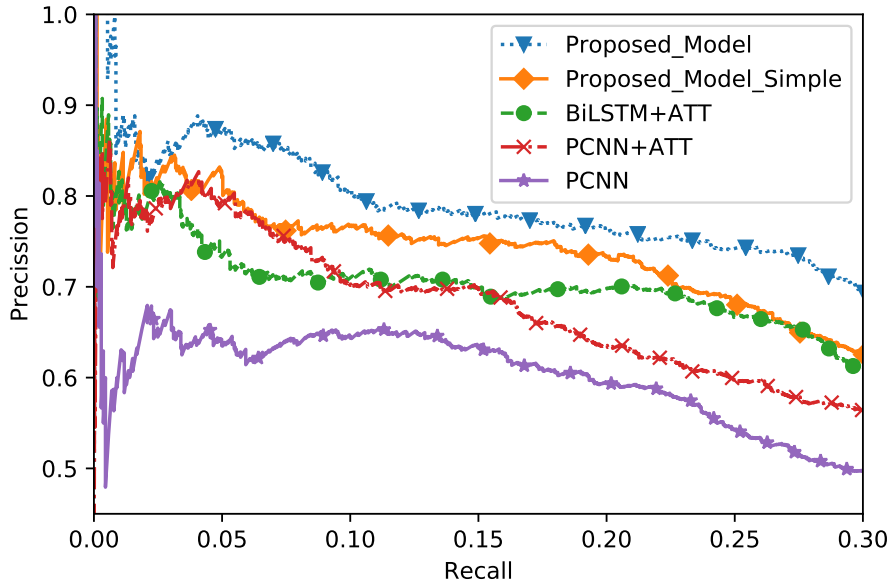

Figure 3.3: The *Precision-Recall* curve for the state-of-the-art and the proposed model.

Table 3.1: *P@N metric for different models.*

P@N(%)	**10%**	**30%**	**50%**
PCNN	69.2	55.5	46.1
PCNN+ATT	77.1	62.2	52.9
BiLSTM+ATT	79.4	68.6	59.3
Proposed-Model-Simple	81.3	70.3	63.6
Proposed-Model	**84.4**	**73.6**	**66.8**

3.5.5 Analysis.

Figure 3.3 includes *Precision-Recall* curves obtained for different methods. As it can be seen, **Proposed-Model-Simple** has higher values of *Precision* for the entire range of *Recall* values compared to the previous state-of-art neural models. Although **Proposed-Model-Simple** and **BiLSTM+ATT** use the same neural model structure and utilize attention mechanisms, the **Proposed-Model-Simple** significantly outperforms the latter one. This proves that removing of noisy and irrelevant tokens from sentences, using the proposed pruning technique, improves the RE task. However, **Proposed-Model-Simple** performs worse compared to **Proposed-Model** and has lower *Precision* values for the entire range of *Recall* values. This shows that incorporating head and tail type information further improves the results.

Also, as seen in Figure 3.3, **PCNN** has the lowest *Precision* values for the entire range of *Recall*, compared to the other models. **PCNN** selects one sentence from each bag as a representation for the entire bag and does not apply sentence-level attention mechanisms to the sentences. It leads to the loss of important information. This can be prevented using attention, especially for the NYT dataset which is a weakly supervised dataset that contains noisy information. Thus, **PCNN+ATT** performs better compared to **PCNN**.

Table 3.1 reports the *P@N* metric for different models. As it is shown, the model proposed in this chapter achieves higher P@N which indicates its efficiency.

3.6 Conclusion

In this chapter, we propose and describe a new model for the RE task. We use dependency trees to introduce a new filtering technique that removes noisy tokens from sentences and keeps the most relevant ones for the RE task. Noisy tokens are words that have no impact on prediction of relations linking head and tail entities in sentences. We also utilize entity type information. As each relation imposes con-

straints on the type of its head and tail entities, the inclusion of this information leads to an actual improvement of the RE task. The experimental results obtained using the NYT dataset indicate that our model outperforms other state-of-the-art neural models and achieves higher values of *Precision* over the entire range of *Recall* values.

Chapter 4

Triple Extraction using Sequence to Sequence Transformers

Extracting relational facts from unstructured text is a crucial task in natural language processing used in many applications, particularly the construction of knowledge graphs. Relational facts are represented as triples in which two entities are connected through a relation. This chapter introduces a new and effective end-to-end method to generate triples from the input text. In the proposed method, we develop an encoder-decoder-based transformer model and warm-start both encoder and decoder with pretrained checkpoints that are publicly accessible. These checkpoints can be taken from different models such as *BERT, GPT-2*, and *RoBERTa*. Experimental results show that our method achieves better results for triple extraction on publicly available datasets (NYT and WebNLG) than the other state-of-the-art techniques.

4.1 Introduction

Triple extraction is an important part of natural language processing (NLP), leading to the construction of knowledge graphs (KGs). Data repositories built based on KGs – called Knowledge Bases (KBs) – such as Freebase [2], DBpedia [3] or Wikidata [4], are well-known resources of relational facts represented as triples [1]. Each triple is a pair of entities connected through a relation. For example, ⟨*Rome, Capital-of, Italy*⟩ is a triple stating the fact that Rome is the capital of Italy. A considerable effort is

being made to derive triples and build KBs using them. The goal of a triple extraction process is to extract triples from sentences automatically.

Previous works are mainly based on the pipeline approach to tackle the triple extraction problem [135–137]. These methods break down the triple extraction into two separate tasks: named entity recognition (NER) and relation classification (RC). First, they extract all possible entities from each sentence and then identify a valid relation for every pair of entities. Their main drawback is that both tasks are related, and performing these tasks independently leads to a propagation of errors occurring in NER to the RC process and its results.

To overcome this problem, a joint extraction of entities and relations is proposed. Some studies introduce feature-based methods [138, 139] while others neural-network-based models [112, 140]. The latter achieve notable results in a joint triple extraction. The authors of [110] have proposed an encoder-decoder method based on a recurrent neural network (RNN). They treat the triple extraction as a sequence to sequence (seq2seq) task [26] and generate triples in an end-to-end manner. Such an approach to the extraction of triples addresses the error propagation problem. Encoder-decoder models based on RNNs are effective models in NLP and have been used in many tasks such as text summarization [141], and machine translation [142]. However, recurrent models suffer from a significant drawback of capturing long-range dependencies. This results in losing important information in the case of long sentences.

In this chapter, we present a transformer based encoder-decoder model for triple extraction. Both encoder and decoder are initialized with pretrained publicly available checkpoints from different models such as: *BERT* [18], *GPT-2* [19, 23] and *RoBERTa* [20, 143]. The applied idea is to treat the process of generating triples as a translation of sentences into simple sentences that represent extracted triples. This means that we convert each triple to a simple sentence that includes the pair of entities and the relation between them for training purposes. For example, the triple ⟨*Rome, Capital-of, Italy*⟩ is converted into the sentence *Rome is the Capital of Italy.* Then our model

is trained to generate such sentences as the output.

Once the simple sentences are extracted, they are converted to triples, further processed, and used to construct a knowledge graph. As a result, we build a graph-based representation of the processed text. Additionally, we obtain information about the strength of the extracted relations that represent the levels of confidence in the relations.

The main contributions of our work are as follows:

- transforming a triple extraction process into a sequence to sequence task where triples of the training data set are transformed into simple sentences;
- utilizing a transformer-based encoder-decoder model as a foundation of the proposed method, and warm-start both encoder and decoder with available pretrained checkpoints taken from different models;
- evaluating our model on two publicly available datasets showing an improved performance when compared with baselines models, what proves the efficiency of our proposed method;

4.2 Related Work

4.2.1 Relational Fact Extraction

Extracting relational triples from unstructured text is essential for building large KGs automatically. Multiple different techniques have been proposed so far.

One of set of methods use pipeline approach [73, 136, 144, 145]. It means the triple extraction task is performed as two sequential subtasks: 1) identification of all entities in a sentence; and 2) relation classification (RC) to determine a valid relation between pair of entities. The pipeline approaches' main disadvantage is ignoring the interrelation between NER and RC, resulting in error propagation.

To address this problem, some researchers have proposed methods that combine

both tasks – extracting entities and extracting relations. Some of them are feature-based techniques [139, 146] that require feature engineering, which makes them inefficient. To mitigate this, neural network-based models have been proposed. They have led to significant improvements. [117] has proposed a reinforcement-learning based method for triple extraction. [109] present a unified tagging scheme by revisiting the triple extraction task and treating it as an end-to-end sequence tagging problem. [147] make use of triplet attention to find the connections between the pair of entities and their related relation. [112] propose a graph convolutional networks (GCNs) model to extract triples while [122] introduce a joint decoding method by presenting a novel cascade binary-tagging setup.

The problem of overlapping entities has been addressed by proposing an end-to-end model [110]. The model uses CopyRE to generate triples from the input text. This has been the first work that converts the triple extraction task into a sequence to sequence task in a generative manner. After that, some improvements in the efficiency of CopyRE have been proposed. [111] introduce multi-task learning equipped with CopyRE to address the weakness of CopyRE. [124] present a contrastive generative transformer (CGT) to ensure that generated triples are faithful. They utilize UniLM-base-uncased model [148] as their architecture.

4.2.2 Text Generation

Recently, many researchers have utilized transformers for Natural Language Understanding (NLU) and language modeling. Transformers have been first introduced in [17] – they have presented an encoder-decoder-based transformer for machine translation. *GPT-2* [19, 23] is a decoder only transformer which is used for text generation while *BERT* [18] is an encoder only transformer used to encode the text representation for NLU tasks. Some works have been presented to enable BERT for sequence generation. [149] introduce Conditional Masked Language Modeling (C-MLM) to fine-tune *BERT* for text generation tasks. In [150], *BERT* has been reformulated as

a Markov Random Field language model, and the first findings on a text generation with increased variety are shown. [151] propose Masked seq2seq (MASS) model for text generation, and the results are meaningful on tasks of unsupervised machine translation and text summarization.

4.3 Problem Statement

In this section, we provide a brief description of our proposed model and its various components.

4.3.1 Objective

The main goal of this chapter is to introduce a novel method for extracting all possible triples from sentences. In order to better procure the correlation between NER and RC, we extract entities and their corresponding relations during the same process, i.e., in a joint manner. Hence, we transform the triple extraction task into a sequence-to-sequence (seq2seq) problem. In other words, we treat the process of extracting triples as translating input sentences into simple sentences that contain triples representing relations between entities found in the input sentences.

In the first step to train a sequence-to-sequence model, we construct output sentences. It means we convert each triple existing in the input sequence into a short sentence and then concatenate these short sentences together via the conjunctive word ***and***.

To illustrate this idea, we provide a simple example, Table 4.1. It includes the *Input Sentence* and two 'gold' triples that are embedded in this sentence. Each triple is converted into its equivalent *t-Sentence*, and both *t-Sentences* are concatenated into – what we call – *Label_Sequence*. Additionally, two tokens *Beginning Of Sentence [BOS]* and *End Of Sentence [EOS]* are added to the beginning and end of both input and output sequences.

Table 4.1: Generation of a *Label_Sequence* for the training phase based on two *Gold Triples* embedded in the *Input_Sequence*.

Input_Sequence : [BOS] Alan Bean, who was part of Apollo 12, was born in Wheeler, Texas. [EOS]
Gold_Triple 1 **:** ⟨ *Alan Bean, was a crew member of, Apollo 12* ⟩ $\xrightarrow[\text{t-Sentence}]{\text{conversion to}}$ "Alan Bean was a crew member of Apollo 12"
Gold_Triple 2 **:** ⟨ *Alan Bean, BirthPlace, Texas* ⟩ $\xrightarrow[\text{t-Sentence}]{\text{conversion to}}$ "Alan Bean was born in Texas"
Label_Sequence : [BOS] Alan Bean was a crew member of Apollo 12 **and** Alan Bean was born in Texas [EOS]

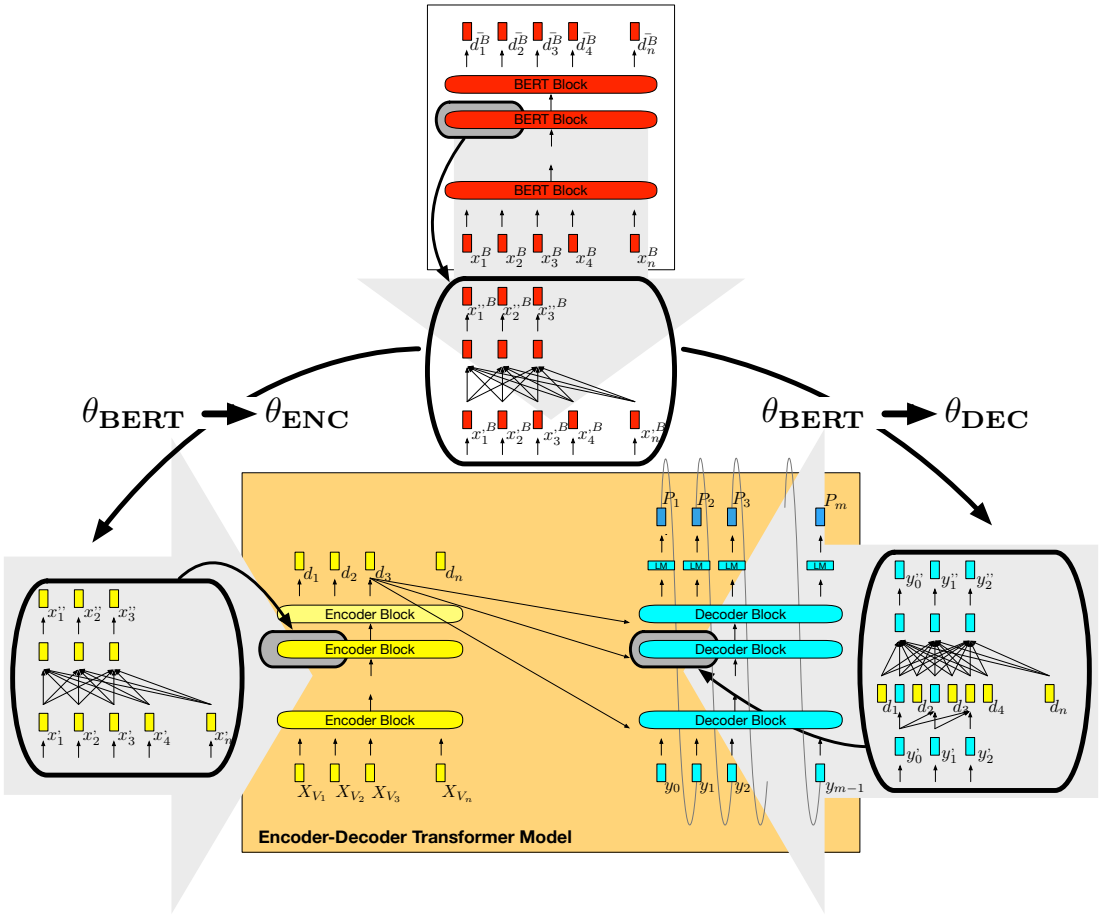

Figure 4.1: Initializing encoder and decoder using the BERT's pretrained checkpoint.

4.3.2 Model Overview

The process of constructing our proposed sequence-to-sequence model requires different phases and components. Let us provide their short introductions.

Conversion of Triples into Sentences.

In order to train our model, we convert 'gold' triples of input sentences into simple sentences that include the pairs of entities and relations between them. Details in Section 4.4.1.

Input Preparation.

We tokenize each sequence, both input and constructed one, using an appropriate tokenizer. The type of tokenizer depends on the checkpoint used to initialize both encoder and decoder.

Model and Its Initialization.

Models built only with encoder or decoder are not suitable for seq2seq tasks. Encoder-only models, like *BERT* [18], need to know the length of the output sequence in advance, and thereby they are not suitable for the triple generation task. Also, although decoder-only models, like GPT-2 [26], perform very well in language generation but they are not well suited for the conditional generation, which limits their application for the triple generation task.

In this work, we employ the seq2seq architecture that is based on an encoder-decoder transformer, Figure 4.1. Seq2seq models have been proved to function better on text generation tasks. Both encoder and decoder are built of transformer layers [17]. We initialize parameter weights of both encoder and decoder using pretrained checkpoints taken from different models such as *GPT-2, BERT*, and *RoBERTa*. The encoder will encode the input sequence into a contextual representation vector, and the decoder will try to generate all possible triples existing in the input sequence in an auto-regressive manner. We will investigate different combinations of checkpoints for warm-starting both encoder and decoders to see which combination performs the best for the triple extraction task.

Algorithm 2 Converting Triples into Label_Sequences

Input:
Set of Gold_Triples occurring in Input_Sequences:
S_{Tr}=$\{(h_1, r_1, t_1), \ldots, (h_n, r_n, t_n)\}$
LookUp_Table:
$\{\langle r_j, vp_j \rangle \; for \; j = 1, \ldots, number\_of\_relations \; \}$

Output:
Set of Label_Sequence: S_{LS}

$S_{LS} \leftarrow \emptyset$
for each $tr_i \in S_{Tr}$ **do**
    $e_1 \leftarrow extract\_head\_ent(tr_i)$
    $e_2 \leftarrow extract\_tail\_ent(tr_i)$
    $r_i \leftarrow extract\_rel(tr_i)$
    $vp_{r_i} \leftarrow LookUp\_Table(r_i)$
    $S_{tr_i} \leftarrow create\_label\_seq(e_1, vp_{r_i}, e_2)$
    if $tr_i = tr_n$ **then**
        $S_{LS} \leftarrow S_{LS} \cup S_{tr_i}$
    else
        $S_{LS} \leftarrow S_{LS} \cup concatenate\_label\_seq(S_{tr_i}, and)$

4.4 Methodology

This section describes the proposed model in detail. First, we discuss how to convert 'gold' triples into simple sentences – *Label_Sequences*. Second, we show how the input and label sequences are tokenized and what kind of tokenizer is used in the encoder and decoder. Third, we investigate the proposed transformer based encoder-decoder model for the triple extraction task. Also, we discuss the effect of warm-starting both encoder and decoder with pretrained checkpoints.

4.4.1 Converting Triples into Sentences

The essential step of the proposed approach is the processing of training data. It means the generation of simple sentences, called *Label_Sequences*, from 'gold' triples that are embedded in the sentences – *Input_Sequences* – of the training data.

Table 4.2: Sample of relations from the NYT dataset and their corresponding verbal phrases.

Header(s) – Relation (r)	Verb Phrase (vp_r)
\people\person – place-of-birth	was born in
\people\person – nationality	is the nationality of
\business\company – place-founded	was founded in
\location\neighborhood – neighborhood-of	is in the neighborhood of

As the first step, we construct a lookup table containing pairs: relation – verbal phrase (vp). It means we assign to each relation that appears in the sentences of the training dataset a single verbal phrase (vp). The vp's are chosen in such a way that they transfer the meaning of the relation embedded in the input sentences. Table 4.2 shows a fragment of the lookup table with a few examples of the relations from NYT dataset and the vp's assigned to them.

For converting triples $\langle h_i, r_i, t_i \rangle$ – *gold triples* – associated with an *Input Sequence (IS)* into a sentence, we execute the following process, Algorithm 2. We extract the vp corresponding to the r_i from the lookup table; we attach h_i and t_i before and after vp. By doing this, a sentence is generated – we call it *t-Sentence* – that is the counterpart sentence for the 'gold' triple. In a case, there is only one triple this becomes *Label_Sequence*. In a case there are more than one 'gold' triple in the input sequence, we create a *t-Sentence* for each triple, and then we concatenate all constructed *t-Sentences* together via a conjunctive word ***and*** to form a single final sentence – called *Label_Sequence*.

Additionally, we add special tokens $[BOS]$ and $[EOS]$ to mark the start and end of the sentence. Such created a set of label sequences constitute a training dataset fed to the model.

4.4.2 Input Encoder

We tokenize input and label sequences using different tokenizers. The selection of the tokenizer depends on the model we use in our encoder-decoder architecture. For example, if the encoder is initialized using *BERT* pretrained checkpoint, then the input sequence is tokenized using WordPiece [152]. On the other hand, if the decoder is initialized using *GPT-2* checkpoint, then the label sequences are tokenized by Byte Pair Encoding (BPE) [41], which matches the *GPT-2* vocabulary. In the case *RoBERTa* checkpoint is used for initializing, BPE is used as a tokenizer. Therefore, the type of tokenizer used for input and label sequences depends on the pretrained checkpoint utilized for initializing the encoder and decoder, respectively.

4.5 Model Architecture

The details of the architecture of our model including its initialization approaches are presented here.

4.5.1 Seq2seq Architecture

The backbone of our encoder-decoder architecture is the model presented in [17] which is an encoder-decoder based transformer model. Given the tokenized input sequence $X = \{x_1, x_2, ..., x_n\}$, the encoder encodes each token into a contextualized vector representation $D = \{d_1, d_2, ..., d_n\}$. The decoder exploits D to generate the output sequence $Y = \{y_1, y_2, ..., y_m\}$, in an auto regressive manner. This can be written in mathematical way using the Bayes' rule as follows:

$$\begin{gathered} p_{\theta_{enc,dec}}(Y|X) = p_{\theta_{dec}}(Y|D) = \prod_{i=1}^{m} p_{\theta_{dec}}(y_i|Y_{0:i-1}, D) \\ where: D = f_{\theta_{enc}}(X) \end{gathered} \tag{4.1}$$

Therefore, the conditional probability of generating a word in each step can be

obtained by applying softmax over the output of the top-most layer – logit vector P:

$$p_{\theta_{dec}}(y_i|Y_{0:i-1}, D) = Softmax(P_i) \tag{4.2}$$

Encoder and decoder are both composed of stacked transformer layers.

Tokens of the input sequence X are first embedded into vectors $X_{V_{(0)}} = \{X_{V_1}, X_{V_2}, ..., X_{V_n}\}$, by summation of token embedding and position embedding together. Afterward, $X_{V_{(0)}}$ passes through transformer layers one by one to be encoded into a contextualized representation vector D:

$$X_{V_{(j+1)}} = transformer(X_{V_{(j)}}) \tag{4.3}$$

In each transformer block, bi-directional self-attention is applied to the input sequence. For this, three matrices are created from each block's input. They are called: query Q, key K, and value V. These matrices are calculated by multiplying X_V by query weight matrices, key weight matrices, and value weight matrices (W^Q,W^K,W^V) respectively.

$$\begin{gathered} Q_{j+1} = X_{V_j} W^Q_{j+1}, \quad K_{j+1} = X_{V_j} W^K_{j+1}, \\ V_{j+1} = X_{V_j} W^V_{j+1} \end{gathered} \tag{4.4}$$

$$Z_{j+1} = Softmax(\frac{Q_{j+1} K^T_{j+1}}{\sqrt{d_k}}) V_{j+1} \tag{4.5}$$

where Z_{j+1} is output of the self attention layer in the block $j+1$ and d_k is the dimension size of K_{j+1}. In each transformer block, there are multiple attention heads that improve the model ability to focus on different positions. Each head has separate weight matrices for query, key and value what leads to a different Q,K and V matrices in each head.

An architecture of the decoder is also made of stacked transformer layers with some differences when compared to the encoder:

1. Encoder's bidirectional self attention layers should be changed to unidirectional

self attention (casual attention) to be fit with auto-regressive style of the decoder. So, Eq. 4.5 is rewritten as follows:

$$Z_{j+1} = Softmax(\frac{Q_{j+1}K_{j+1}^{T}}{\sqrt{d_k}} + M)V_{j+1} \tag{4.6}$$

where M is a masking matrix: it is 0 for the left side context to allow them to attend the attention, and it is $-\infty$ for right side context to prevent them from attending the attention.

2. Decoder utilizes the encoder's final block output (contextualized encoded sequence, D) to generate output at each time step. This is obtained by adding a cross attention layer in the decoder.

3. On top of the final decoder's block, the Language Model Head (LM Head) layer is added to generate a token with the highest probability at each time step.

4.5.2 Applying Pretrained Checkpoints

The architecture of the proposed model is the same as *BERT-base* that is slightly different from the vanilla transformer employed in [17]. BERT uses GELU [153] as an activation function other than RELU. Also, *BERT-base* has 12 transformer layers compared to [17], which has just 6 layers.

In this work, transformer layers are warm-started using freely obtainable pretrained checkpoints [143]. These checkpoints can be taken from different models like: *BERT, GPT-2* and *RoBERTa*. For example, the encoder is initialized using RoBERTa, while GPT-2 checkpoints is used to initialize the decoder. In general, both encoder and decoder can be initialized with 'any combination' of checkpoints.

For injecting a pretrained checkpoint into an encoder-decoder-based transformer model, all the model's parameters are initialized with the parameters corresponding to the checkpoint. Let us say we use *BERT pretrained checkpoint* to initialize both encoder and decoder as illustrated in Figure 4.1. The following steps should be followed:

- **Encoder Initialization:** For the encoder, we compare *BERT*'s architecture to the encoder's one to identify similarities and differences. Any layer in the encoder that exists in *BERT* is initialized using *BERT*'s counterpart pretrained weight parameters. However, if there is a layer in the encoder that does not have its equivalent in the *BERT*'s configuration, it is initialized randomly. Because the configuration of the transformer used in this work is the same as BERT, therefore every encoder layer can be initialized using *BERT*'s equivalent pretrained weights – so, there is no randomly initialized parameter in the encoder.

- **Decoder Initialization:** *BERT* is an encoder-only transformer. If we want to initialize the decoder with the *BERT*'s pretrained checkpoint, some modifications have to be performed to make the *BERT*'s architecture compatible with the decoder's one. The modifications are:

    1. *BERT*'s self attention layer is bidirectional. This should be changed to uni-directional to match the auto-regressive fashion of the decoder.

    2. *BERT* does not have a cross attention layer. Yet, the decoder requires this layer as it is conditioned on the last encoder's block output (D). So, a layer should be added to the *BERT*'s structure right after the self-attention layer. This layer's weights are initialized randomly.

    3. *BERT* does not have a language model head layer on top of its last encoder block. However, as the outputs of the decoder are logit vectors and the most probable word should be selected from these vectors at each time step, the decoder needs this layer. So, again, a layer should be added to the *BERT*'s architecture. Its weights are initialized with the *BERT*'s word embedding matrix, so they are not initialized randomly.

  Once the above modifications are performed, both *BERT*'s and decoder's archi-

tectures are identical, and we warm-start the decoder with *BERT* checkpoint.

4.6 Experiments

4.6.1 Dataset

For evaluation purposes, we will utilize two widely used datasets for evaluating our model: New York Times (NYT) [126], and WebNLG [154]. NYT dataset was built using the distant supervision method and used for relation and triple extraction tasks. This dataset contains 56195 sentences for training, 5000 sentences for testing, and 5000 sentences for validation. Also, it has 24 predefined relations. WebNLG dataset was initially used for a natural language generation (NLG) task. It consists of 5019 sentences for training, 703 sentences for testing and 500 sentences for validation. This dataset has 246 predefined relations.

4.6.2 Settings

The encoder-decoder-based transformer architecture is warm-started by pretrained checkpoints. All applied checkpoints correspond to the *base* architecture – it means the one that has 12 layers, 12 attention heads, a filter size of 3072, and a hidden size of 768. Therefore, whenever *BERT* pretrained checkpoint is mentioned in the experiments, that relates to *BERT-Base Cased* configuration.

All models are trained for 60 epochs with the batch size of 32 for NYT dataset, and 100 epochs and the batch size of 8 for WebNLG dataset. Adam optimizer is used with the learning rate of $5 * 10^{-}5$, and the beam size is set to 4.

We run our experiments on two *NVIDIA TITAN RTX* GPUs. All the hyperparameters are fine-tuned carefully using the validation dataset.

4.6.3 Baselines and Analysis

In this section, we compare the performance of our proposed model with other state-of-the-art models, such as **NovelTagging** [109], **CASREL** [122] and **MrMep** [147]

that are categorized as *extractive* methods, and **CGT** [124], **CopyMTL** [111], and **CopyRE** [110] that are *generative* methods.

Our model is a *generative* method with an architecture that is an encoder-decoder-based transformer. Both encoder and decoder are initialized with pretrained publicly available checkpoints obtained from different models. We will evaluate different scenarios.

RND2RND – both encoder and decoder are initialized randomly without using any pretrained checkpoints.

BERT2RND – the encoder is initialized with *BERT* checkpoint, while the decoder is initialized randomly. In this case, the tokenizer and the vocabulary used for the decoder's sequences (label sequences) are the same as of *BERT-base Cased*.

RND2BERT – the encoder is initialized randomly, and the decoder is initialized using *BERT* checkpoint. In this case, the tokenizer and the vocabulary used for the encoder's sequences (input sequences) are the same as *BERT-base Cased*.

RND2GPT – the encoder is initialized randomly, while the decoder is initialized using the *GP2-2* checkpoint.

RoBERTa2GPT – the encoder is initialized with the *RoBERTs* checkpoint, while the decoder is initialized using the *GPT-2* checkpoint.

BERT2BERT – both encoder and decoder are initialized by the *BERT* checkpoint.

BERT2GPT – the encoder is initialized with the *BERT* checkpoint, yet the decoder is initialized by the *GPT-2* checkpoint.

BERTShared – this one is the same as *BERT2BERT*, but the parameters of encoder and decoder are tied together and shared. This reduces the overall parameters of the model and thereby requires less memory.

RoBERTaShared – both encoder and decoder are initialized using the *RoBERTa* checkpoint, and parameters of encoder and decoder are tied and shared.

We use three matrices for the evaluation: Precision (P), Recall (R), and $F1$ score

(F). We report their values for our method and the baseline methods.

4.6.4 Main Results

The obtained values of P, R, and $F1$ are presented in Table 4.3. As we can see, the usage of a warm-starting encoder and decoder with pretrained checkpoints improves the performance of the triple extraction task substantially. For both NYT and WebNLG datasets, our model outperforms other state-of-the-art generative methods. For example, for WebNLG dataset, when *RoBERTaShared* is utilized, our obtained value of *F1* is: **2.5%** higher than the one obtained for *CGT*; **48.8%** more than of *CopyRE*; and **29.5%** higher than compared to *CopyMTL*. Moreover, our model's performance on WebNLG dataset is proportional to extractive methods like *CASREL*.

In the case of NYT dataset, our model provides better *F1* score against all generative and extractive methods. The *RoBERTaShared*'s model has the *F1* score higher by **1.4%** when compared to the values obtained for *CGT*, and higher by **0.9%** when compared with the value for *CASREL*.

The main characteristic of our model that distinguishes it from other baselines is the high scores of recall (R) without significant degradation of the value of precision. As it can be seen in Table 4.3, our model achieves significantly higher values of recall for both datasets.

4.6.5 Initialization Scenarios of Encoder and Decoder

Encoder and decoder can be initialized in different ways. Depending on how they are initialized, we see variations in the models' performance, Table 4.3. For instance, *BERT2RND* outperforms *RND2RND* by a large margin, yet *RND2BERT* has results comparable to *RND2RND*. It indicates that the warm-starting of the encoder is a key factor in improving the model's performance. However, the warm-starting of the decoder alone has no significant impact on the results.

Our decoder's architecture is the same as of GPT-2. Therefore, the initial esti-

Table 4.3: Values of precision (P), recall (R), and *F1*-score for different models.

Model		NYT			WebNLG		
Model Type	**Model Name**	**P**	**R**	**F**	**P**	**R**	**F**
Extractive Methods	NovelTagging	61.5	41.4	49.5	-	-	-
	CASREL	89.7	89.5	89.6	93.4	90.1	91.8
	MrMep	77.9	76.6	77.1	69.4	77.0	73.0
Generative Methods	CopyMTL	75.7	68.7	72.0	58	54.9	56.4
	CopyRE	61.0	56.6	58.7	37.7	36.4	37.1
	CGT	94.7	84.2	89.1	92.9	75.6	83.4
Proposed Method	RND2RND	82.0	76.5	79.1	80.6	78.0	79.3
	RND2GPT	77.0	72.1	74.5	65.5	65.6	65.6
	RND2BERT	79.6	73.8	76.6	79.0	77.0	78.0
	BERT2RND	88.8	86.4	87.5	85.1	84.4	84.7
	BERT2GPT	86.5	85.8	86.1	84.6	83.1	83.8
	RoBERTa2GPT	89.2	88.7	88.9	85.2	83.0	84.1
	BERT2BERT	90.3	88.4	89.4	84.8	84.7	84.8
	BERTShared	90.5	89.7	90.1	85.6	85.0	85.3
	RoBERTaShared	91.0	90.0	**90.5**	85.5	86.3	**85.9**

mate is that the warm-starting of the decoder with GPT-2 should give better results than warm-starting with BERT. It could be because we have modified the BERT's structure to fit with the decoder. Despite that, we see quite a different situation, Table 4.3. Results of the model with *BERT2BERT* and *RND2BERT* initialization have gained better outcome compared to *BERT2GPT* and *RND2GPT* respectively.

Further analysis of results shows a good performance of 'shared' models, i.e., *BERTShared* and *RoBERTaShared*. These models perform better on both datasets in the comparison to non-shared models. For instance, *BERTShared* outperforms

Table 4.4: Performance of model trained on different size of data: *RoBERTaShared* as initializer.

Reduction Size	NYT			WebNLG		
Training Dataset Reduced by: (%)	P	R	F	P	R	F
0	91.0	90.0	90.5	85.5	86.3	85.9
25	89.5	89.5	89.5	76.1	83.2	79.5
50	86.5	88.1	87.3	72.2	77.7	74.8
75	80.6	81.9	81.2	57.7	64.1	60.8

BERT2BERT by **0.7%** and *BERT2RND* by **2.6%** in the *F1* score on NYT dataset.

4.6.6 Impact of Dataset Size

We also investigate the impact of the size of the training set on the performance of our model. For this purpose, we reduce the size of both *WebNLG and NYT* training datasets by 0%, 25%, 50%, 75%. For instance, 0% means we do not change the size of the dataset, while 75% means we reduce the size by 75% (use only 25%). The reduction is performed by random sub-sampling of the original dataset. We sub-sample the dataset in a way that the number of instances for each predefined relation is reduced by the same ratio.

The best performing model (*RoBERTaShared*) is used for the experiment, Table 4.4. As it can be seen, by reducing the training set by 75%, the *F1* score is reduced just by 9.3% for NYT dataset and 25.1% for WebNLG dataset. The more significant decrease for WebNLG can be due to the more significant number of predefined relations in the dataset. The results show that even with a smaller size of the training set, our model achieves significant results.

4.6.7 Error Analysis

The fact that the results are very good yet not perfect has triggered a need to analyze the erroneous cases. We show a few examples of triples generated incorrectly by our model. Note: we show triples that have been built using a simple process of converting sentences generated by the proposed model, i.e., *Label_Sequences*, into triples. We have identified four different groups:

1. **Wrong Entity:** the head or tail entities are detected incorrectly by the model. The sample of **Group 1**, Table 4.5, is an illustration of this case. It results from an unclear definition of entity occurring in the gold triple – this entity is expressed implicitly in the input sentence. Therefore, the model fails to generate the valid triple.

2. **Displaced Entity:** similar to the previous error – one of the model's generated entities, head or tail, is different from the entity of gold triple. However, in this case, the generated triple is still a valid triple that can be deduced from the input sequence. For example, let us take a look at the sentence from **Group 2**, Table 4.5. The tail entity for the gold triple is *Indonesia*; however, our model generates *Singapore* instead, which can still be a valid triple generation based on the input sequence. However, as the generated triple differs from the gold triple, we still count this as a wrongly extracted triple.

3. **Dominant Relation:** while generated head and tail entities are the same as in the gold triple, the generated relation is wrong. For the sequence in **Group 3**, Table 4.5, our model generates *'place_lived'* as the relation, however the true relation is *'place_of_birth'*. This error can be due to an unbalanced training set. The number of triples with *'place_lived'* as the relation is much larger than the number of triples with *'place_of_birth'* as the relation. The model is more prone to generate the dominant relation.

4. **Opposite Relation:** the relation generated by our model is the inverse of the one from the gold triple. The example of that is shown in Table 4.5, **Group 4**. As it is seen, the model generates *'followed_by'* instead of *'preceded_by'* as the relation. Nonetheless, the generated triple is equivalent to the gold triple and conveys the same meaning. Despite that, we still count it as a wrong generation as it is not the same as the gold triple.

Table 4.5: Examples of four different groups of errors

Group 1: Buzz Aldrin is a national of the United States whose leader is Joe Biden. He was born in Glen Ridge, Essex County, New Jersey.
Gold_Triple: ⟨ *Biden, leaderName, United States* ⟩
Generated Triple: ⟨ *Biden, leaderName, New Jersey* ⟩
Group 2: Beef kway teow is a dish found in Singapore and Indonesia. Kway teow, beef tender loin, gula Melaka, sliced, dried black beans, garlic, dark soy sauce, lengkuas, oyster sauce, soya sauce, chilli and sesame oil are main ingredients in beef kway teow.
Gold_Triple: ⟨ *Beef kway teow, region, Indonesia* ⟩
Generated Triple ⟨ *Beef kway teow, region, Singapore* ⟩
Group 3: And in fact the Syrian foreign minister, Farouk al-Sharaa, speaking at a news conference in Damascus, also condemned the attack.
Gold_Triple: ⟨ *Farouk al-Sharaa, place_of_birth, Damascus* ⟩
Generated Triple: ⟨ *Farouk al-Sharaa, place_lived, Damascus* ⟩
Group 4: Above the Veil, from Australia, is the third book in a series after Aenir and Castle.
Gold_Triple: ⟨ *Above the Veil, preceded_by, Aenir* ⟩
Generated Triple: ⟨ *Aenir, followed_by, Above the Veil* ⟩

4.7 Conclusion

In this chapter we proposed a novel approach for extracting triples from the text. We have converted the triple extraction problem into a sequence to sequence task. The proposed model is trained on datasets with automatically constructed label sequences. Triples to be extracted from the input sequences are converted into simple sentences. It means our model's task is to generate these sentences as output triples.

The backbone architecture of the model is a transformer-based encoder-decoder. Both encoder and decoder are initialized with publicly available pretrained checkpoints of different models such as *BERT, RoBERTa*, and *GPT-2*. The inclusion of pretrained checkpoints leads to the significant improvement of the triple generation process. We have conducted experiments on two widely used datasets to validate the efficiency of our model. The experimental results confirm that our model achieves state-of-the-art results compared to other baselines. Also, the proposed model provides good results when trained on the reduced datasets.

Chapter 5

Construct Knowledge Graph from Triples

In this chapter we will explain how we can build a knowledge graph out of extracted triples from plain text.

5.1 Introduction

Knowledge graph(KG) is a visualization method that recently has attracted lots of attention for the purpose of visualizing data. The graph-based database, collects data from different types and sources and merges them together through nodes and links. After merging all the data, we end up having a network of entities(nodes) which are linked together via relation(s).

In the chapter 4 we proposed a method a extract triples from plain text. These triples are a pair of entities that are connected together through a relation.However, the question is how to utilize these triples for different purposes. In this chapter we will talk about creating a KG out of extracted triples from WebNLG dataset. The extracted triples are processed and used to build a knowledge graph. Complete control of this process allows determining weights of the relations (triples). The weights reflect the frequency of occurrences facts represented by the relations and provide the degree of confidence in the facts.

5.2 Graph Construction

Triples extracted from a text, considered as a part of a larger task of knowledge extraction, provide useful information that can be applied for variety of purposes. The obtained structured information can be used to enhance recommendation systems [155, 156], question-answering tasks [157], or building more comprehensive reasoning and generative systems [158–161], just to name a few.

In the chapter, we illustrate the utilization of extracted relations to construct a knowledge graph. We build a graph representing a testing set of WebNLG dataset [154].

5.2.1 Process Description

The graph construction process we illustrate here involves several processing steps. All these steps are presented in Algorithm 3. In the beginning, line 2, the generated by our model *Label_Sentences* are converted into triples – an 'inverse' process to the one presented in Section 4.4.1 is applied. Then, a few steps, lines 3 to 11, are performed to identify types of entities that constitute subjects and objects of the triples. For this purpose we use *flairNLP* library[1]. It allows us to identify the following $Entity\_Names$: Buildings, Cardinal, Date, Event, Geographical Location, Location, Organization, Person, Product, Quantity, Work-of-Art, and Other. Once the entities are identified, triples $\langle --, is\_type\_of, Entity\_Name \rangle$ are created and added to the set of all triples.

An essential part of the process is 'name normalization,' lines 12 to 21. Some entities that represent the same thing can be labeled with variations of their names. In most cases, it means shorter or longer names. We propose a straightforward approach to address this issue to normalize them. Once we recognize types of times, we look at the entities of the same type and determine which names are subsets of 'full' names. It means we identify which names are partial ones compared with 'larger'

[1] https://github.com/flairNLP/flair

names. For example, the name 'Alan S' constitutes a part of 'Alan Shepard.' In such a case, we replace a shorter name with a longer one – 'Alan S' is replaced by 'Alan Shepard.' This process is represented by lines 11 to 20, Algorithm 3.

The last stage of the process is dedicated to performing probably the most compelling aspect of the whole graph construction process – assigning weights to triples and importance to nodes. Extraction of triples from a relatively large text means that the same triple can occur/be extracted multiple times. We use this to build a weighted KG. It is important to incorporate information about the frequency of specific facts. We envision this information being interpreted/perceived as confidence in the extracted relations. From a common-sense perspective, if a given fact is mentioned more often, we assume a higher degree of confidence in it. The steps of this process are included in lines 22-33, Algorithm 3.

5.2.2 Constructed Knowledge Graph – Overview

The set of triples obtained from a testing part of *WebNLG* dataset has contained 728 entities and 1554 links between them. Because we perform open-source extraction, we do not have a standardized vocabulary. In total, there are 116 different relations.

The constructed KG is shown in Figure 5.1. We mark the nodes representing types of extracted entities. These nodes are connected, via the relation *is_type_of*, to nodes of extracted triples.

5.2.3 Constructed Knowledge Graph – Zoom-in Sample

To illustrate details of the constructed KG, let us take a look at a fragment of the graph, Figure 5.2. It shows several nodes that are related to the entity *NASA* of the type *ORGANIZATION*. The focus is on the node representing 'Alan Shepard' and its relations with other nodes – all marked in dark blue, and for the illustrative purposes, their thickness represent their importance, i.e., frequency of occurrence. Besides the relation is_type_of, the node 'Alan Shepard' is connected to other nodes via different

Algorithm 3 Processing Triples for Graph Construction

Input:
set of generated Label_Sequences by our method : S_{GenLS}
set of recognizable entities: S_{Entity_Types}

Output:
set of 'normalized' and weighted triples: S_{Tr}^{F}

$S_{Tr} \leftarrow \emptyset$
▷ translating Label_Sequences to triples
$S_{Tr} \leftarrow translate(S_{GenLS})$
▷ entity recognition in text
for each $EnType_k \in S_{Entity\_Types}$ **do**
    create set S_{EnType_k} of entities of type $EnType_k$
▷ construction of triples with is_type_of relation
for each $EnType_k \in S_{Entity\_Types}$ **do**
    for $e_i \in S_{EnType_k}$ **do**
        create triple $< e_i, is\_type\_of, EnType_k >$
        $S_{Tr} \leftarrow$ add newly created triple
▷ entity name normalization
for each $EnType_k \in S_{Entity\_Types}$ **do**
    for $e_i \in S_{EnType_k}$ **do**
        for $e_j \in S_{EnType_k}$ **do**
            if name of e_j part of name of e_i **then**
                remove e_j from S_{EnType_k}
                replace e_j with e_i in all triples in S_{Tr}
▷ adding weights to triples
initialize weight w_i of each triple to 1
$S_{Tr}^{F} \leftarrow \emptyset$
while $S_{Tr} \neq \emptyset$ **do**
    for each t_i from S_{Tr} **do**
        for each t_k from S_{Tr} **do**
            if $t_i == t_k$ **then**
                $w_i += 1$
                remove t_k from S_{Tr}
    remove t_i from S_{Tr}
    $S_{Tr}^{F} \leftarrow t_i$

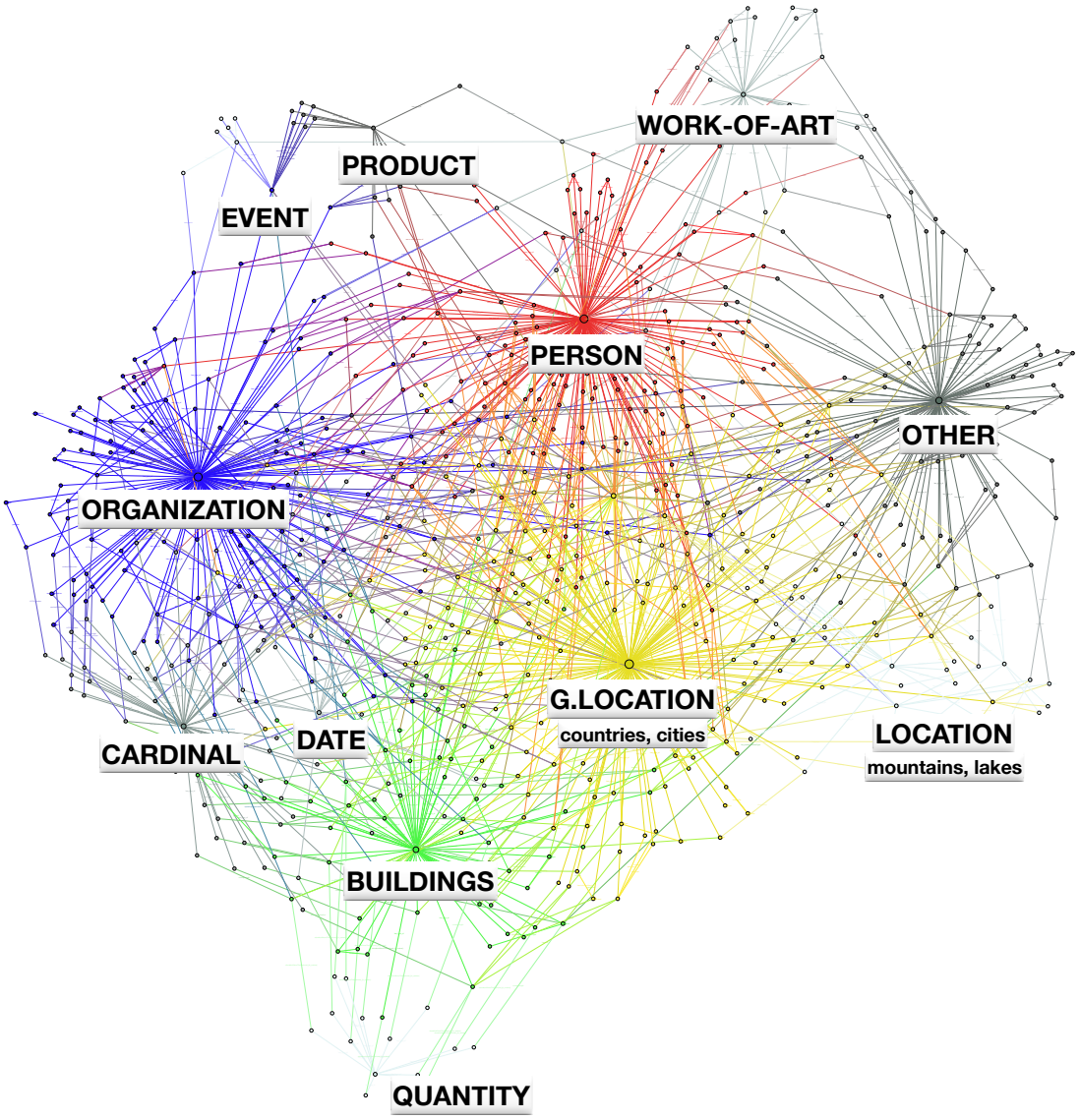

Figure 5.1: Knowledge Graph constructed from extracted triples.

relations extracted from the text. As we see it, the thickness of edges represents the number of occurrences of a given relation. For example, the relations *birthPlace* and *deathPlace* occurred twelve times, the relations as *occupation* and *crew_member* five times, while *selected_by_NASA* and *nationality* four and one, respectively.

As mentioned earlier, these weighted relations allow us to talk about confidence in facts represented by the relations. For example, the snippet of the graph is a clear indication that there is a little doubt in the facts that Alan Shepard's places of birth

and death are New Hampshire and California, respectively.

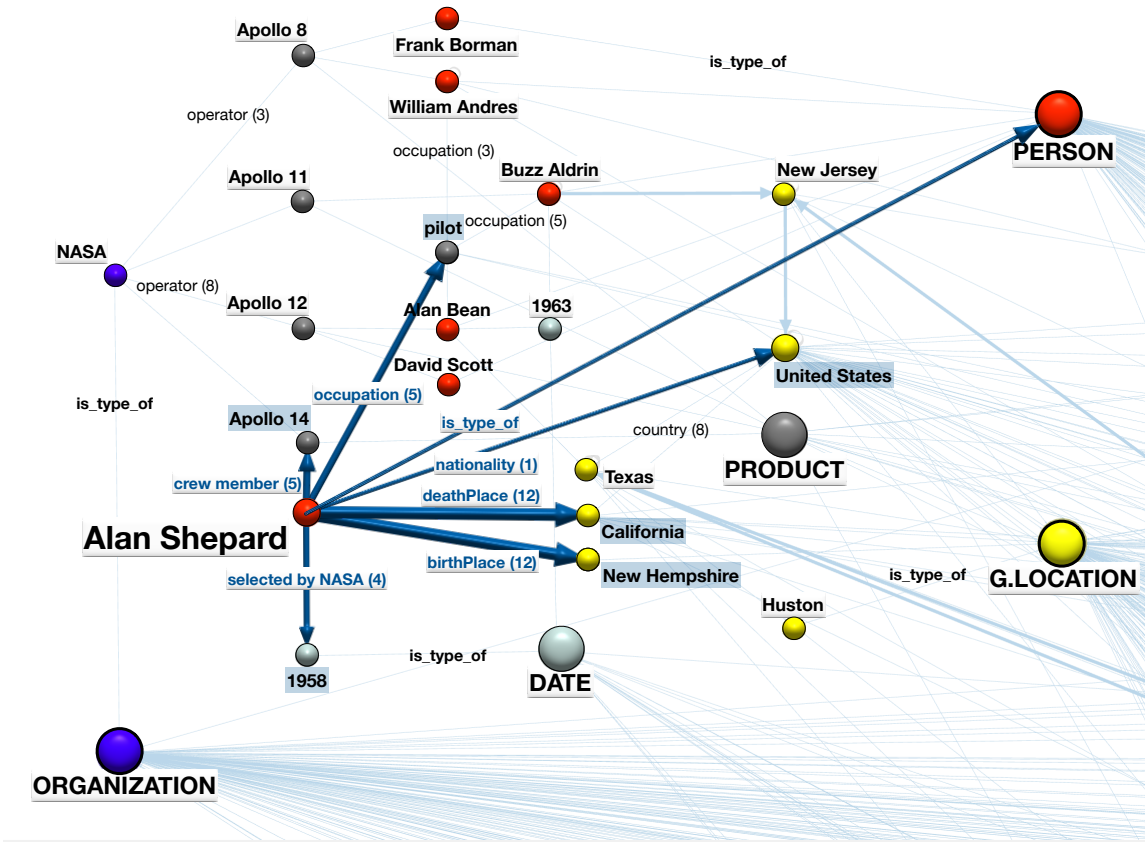

Figure 5.2: Knowledge Graph – fragment – NASA and Apollo missions; the node 'Alan Shepard' and the relations between it and other nodes are emphasized; numbers in brackets represent frequency of occurrences of relations.

5.3 Conclusion

In this chapter we showed a simple process of constructing a weighted knowledge graph based on the triples extracted from a text. Methods for entity recognition, name normalization, and assigning weights to edges are developed. The obtained graph represents the text as a set of triples and provides essential information about the degree of confidence in the extracted facts. Such knowledge can enhance systems requiring semantics and the importance of facts.

Chapter 6

Conclusion

6.1 Summary

A constantly growing amount of textual data can be perceived as a drawback from the users' perspective. It is almost impossible to be up-to-date with generated information. Therefore, the ability to automatically process the data and transform it into a format easy to analyze and digest by machines becomes increasingly essential. The introduction of the Semantic Web has created several ideas, concepts, and solutions that address the issue of intelligent utilization of the web and information stored on it. One of the most appealing contributions of the Semantic Web is the Resource Description Framework.

The Resource Description Framework – RDF – is a graph-based format for representing information. Its basic building block is a triple <*head, relation, tail*>, where *head* and *tail* are two entities – *head* being described by *tail* via a *relation* that links them together. One of the most challenging issues related to RDF utilization is the automatic extraction of triples from textual data.

The main goal of this thesis has been to propose, develop, and validate a comprehensive system for identifying and extracting relational facts from plain text and use these relational facts to create or update knowledge graphs (KGs). This goal has been approached gradually, i.e., starting with a simple problem that focuses only on identifying relations between entities, when *head* and *tail* are already identified in

a sentence, and finishing with extracting all components of triples from a sentence simultaneously.

First, in Chapter 3, we propose a new method for relation extraction (RE) using syntactic information drawn from a sentence dependency tree in addition to information about types of entities. The developed approach analyses a sentence dependency tree and removes noisy tokens. It cleans the input sentences, and only the most important tokens related to RE task are left. Noisy tokens are the words that can be removed from a sentence without affecting the implied relation linking head and tail in the sentence. The approach also uses entity-type information. As each relation imposes restrictions on the type of its head and tail entities, incorporating this information into the model makes an outstanding improvement to the RE task.

A process of extracting a full triple, i.e., all three components of it, has been addressed next, Chapter 4. We propose a new triple extraction method using sequence to sequence transformers in this case. We have converted the triple extraction problem into a sequence-to-sequence translation task. The input to the model is a text sequence, while the output is a sequence representing head, tail, and relations between them. A transformer that includes both encoder and decoder is used to obtain such functionality. Both encoder and decoder are initialized with free available pretrained checkpoints of different models such as *BERT, RoBERTa*, and *GPT-2*.

Finally, in Chapter 5, we demonstrate how a knowledge graph can be built using relational facts extracted from the text. The developed simple methodology that uses sequences generated with our transformer-based method from a plain text has been applied to construct a knowledge graph with human supervision.

6.2 Contributions

The list of original accomplishments described in this thesis can be summarized as the following:

- Proposing a new technique for relation extraction (RE) task using dependency tree and entity type information. We apply LSTM networks with an attention mechanism to classify the true relation in each sentence. The experimental results obtained using the NYT dataset indicate that our model outperforms other state-of-the-art neural models and achieves higher values of *precision* over the entire range of *recall* values.

- Proposing a novel approach for the triple extraction task. We utilize transformer-based encoder-decoder models to generate the triples. Both encode and decoder are initialized using pretrained checkpoints from different models. Applying pretrained checkpoints on the encoder and decoder leads to a significant performance boost in the triple generation task. We have performed experiments on two widely used datasets: WebNLG and NYT, to demonstrate the efficiency of our model. The experimental results show that our model achieves state-of-the-art results compared to other baselines. We have also tested the model with a reduced dataset size to see its impact on the model. The results confirm the efficiency of our model even with a reduced dataset size.

- Proposing a procedure to construct a knowledge graph based on the extracted triples from a text. We used the triples extracted from the WebNLG dataset and created a weighted knowledge graph. This process is fully automated, and there is no need for a human to extract the triples or build the graph. The proposed method also provides information about the strength of relations between entities. It is determined based on the frequency of occurrences of triples extracted from the text.

6.3 Future Considerations

In general, the methods proposed in this thesis build a fundamental structure for future research related to knowledge extraction and construction of knowledge graphs.

Based on the proposed ideas, one can continue working on information extraction topic, considering the followings:

- We utilized recurrent neural networks for a relation extraction task. However, nowadays, transformers are proven to be very useful in many NLP tasks. So, we can utilize transformer structures and fine-tune them for relation extraction.

- We treated a triple extraction task as a sequence-to-sequence task. We used the sequence to sequence transformers and initialized them with pretrained checkpoints from different models. However, the T5 model has shown prominent results in many NLP tasks. For future work, we can investigate the utilization of T5 for triple extraction and compare the results with those reported in the thesis.

- We tested our triple extraction method on WebNLG and NYT datasets. However, it will be important and beneficial to have other supervised datasets related to the task of triple extraction. It would enable more experiments and improve the evaluation strategies. It would be interesting to apply our method on datasets representing specialized domains such as medical ones.

- We worked with a few datasets. However, creating a supervised dataset for the triple extraction task is time-consuming and expensive. To overcome this issue, we can investigate few-shot learner methods for triple and relation extraction. It would decrease the need for large sizes of training data.

Bibliography

[1] S. Powers, *Practical RDF: solving problems with the resource description framework.* " O'Reilly Media, Inc.", 2003.

[2] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD Int. Conf. on Management of data*, 2008, pp. 1247–1250.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*, Springer, 2007, pp. 722–735.

[4] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Communications of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[5] M. Yu, W. Yin, K. S. Hasan, C. d. Santos, B. Xiang, and B. Zhou, "Improved neural relation detection for knowledge base question answering," *arXiv preprint arXiv:1704.06194*, 2017.

[6] F. O. Isinkaye, Y. O. Folajimi, and B. A. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian informatics journal*, vol. 16, no. 3, pp. 261–273, 2015.

[7] C. Quirk and H. Poon, "Distant supervision for relation extraction beyond the sentence boundary," *arXiv preprint arXiv:1609.04873*, 2016.

[8] M. Boden, "A guide to recurrent neural networks and backpropagation," *the Dallas project*, 2002.

[9] D. Zhang and D. Wang, "Relation classification via recurrent neural network," *arXiv preprint arXiv:1508.01006*, 2015.

[10] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] Y. Xu, L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin, "Classifying relations via long short term memory networks along shortest dependency paths," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1785–1794.

[13] D. Jurafsky and J. H. Martin, *Speech and language processing. vol. 3*, 2014.

[14] T. Mikolov, K. Chen, G. Corrado, J. Dean, L Sutskever, and G Zweig, "Word2vec," *URL https://code. google. com/p/word2vec*, vol. 22, 2013.

[15] X. Rong, "Word2vec parameter learning explained," *arXiv preprint arXiv:1411. 2738*, 2014.

[16] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[19] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[20] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[23] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[24] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[25] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[26] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

[27] Z. Fan, Y. Gong, D. Liu, Z. Wei, S. Wang, J. Jiao, N. Duan, R. Zhang, and X. Huang, "Mask attention networks: Rethinking and strengthen transformer," *arXiv preprint arXiv:2103.13597*, 2021.

[28] S. Mehta, M. Ghazvininejad, S. Iyer, L. Zettlemoyer, and H. Hajishirzi, "Delight: Very deep and light-weight transformer," 2020.

[29] H. Yan, B. Deng, X. Li, and X. Qiu, "Tener: Adapting transformer encoder for named entity recognition," *arXiv preprint arXiv:1911.04474*, 2019.

[30] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[31] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient content-based sparse attention with routing transformers," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 53–68, 2021.

[32] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[33] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *International Conference on Machine Learning*, PMLR, 2020, pp. 1691–1703.

[34] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[35] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer," *arXiv preprint arXiv:1809.04281*, 2018.

[36] L. Dong, S. Xu, and B. Xu, "Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2018, pp. 5884–5888.

[37] X. Chen, Y. Wu, Z. Wang, S. Liu, and J. Li, "Developing real-time streaming transformer transducer for speech recognition on large-scale dataset," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2021, pp. 5904–5908.

[38] W. Yu, J. Zhou, H. Wang, and L. Tao, "Setransformer: Speech enhancement transformer," *Cognitive Computation*, pp. 1–7, 2021.

[39] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[40] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[41] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[42] A. Conneau and G. Lample, "Cross-lingual language model pretraining," *Advances in neural information processing systems*, vol. 32, 2019.

[43] H. Le, L. Vial, J. Frej, V. Segonne, M. Coavoux, B. Lecouteux, A. Allauzen, B. Crabbé, L. Besacier, and D. Schwab, "Flaubert: Unsupervised language model pre-training for french," *arXiv preprint arXiv:1912.05372*, 2019.

[44] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," To appear, 2017.

[45] M. Schuster and K. Nakajima, "Japanese and korean voice search," in *2012 IEEE Inter. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2012, pp. 5149–5152.

[46] T. Kudo, "Subword regularization: Improving neural network translation models with multiple subword candidates," *arXiv preprint arXiv:1804.10959*, 2018.

[47] J. L. Martinez-Rodriguez, I. López-Arévalo, and A. B. Rios-Alvarado, "Openie-based approach for knowledge graph construction from text," *Expert Systems with Applications*, vol. 113, pp. 339–355, 2018.

[48] E. Miller, R. Swick, and D. Brickley, *resource description framework"-http://www. w3. org*, 2004.

[49] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, *et al.*, "Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic web*, vol. 6, no. 2, pp. 167–195, 2015.

[50] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 697–706.

[51] C. Fellbaum, "Wordnet," in *Theory and applications of ontology: computer applications*, Springer, 2010, pp. 231–243.

[52] R. Volz, J. Kleb, and W. Mueller, "Towards ontology-based disambiguation of geographical identifiers.," in *I3*, 2007.

[53] D. Vrandečić, "Wikidata: A new platform for collaborative data collection," in *Proceedings of the 21st international conference on world wide web*, 2012, pp. 1063–1064.

[54] N. Chah, "Freebase-triples: A methodology for processing the freebase data dumps," *arXiv preprint arXiv:1712.08707*, 2017.

[55] L. Hirschman and R. Gaizauskas, "Natural language question answering: The view from here," *natural language engineering*, vol. 7, no. 4, pp. 275–300, 2001.

[56] J. Zhang, C. Zong, *et al.*, "Deep neural networks in machine translation: An overview.," *IEEE Intell. Syst.*, vol. 30, no. 5, pp. 16–25, 2015.

[57] J. Zhang and N. M. El-Gohary, "Semantic nlp-based information extraction from construction regulatory documents for automated compliance checking," *Journal of Computing in Civil Engineering*, vol. 30, no. 2, p. 04 015 014, 2016.

[58] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[59] A. Mansouri, L. S. Affendey, and A. Mamat, "Named entity recognition approaches," *International Journal of Computer Science and Network Security*, vol. 8, no. 2, pp. 339–344, 2008.

[60] M. Marrero, J. Urbano, S. Sánchez-Cuadrado, J. Morato, and J. M. Gómez-Berbís, "Named entity recognition: Fallacies, challenges and opportunities," *Computer Standards & Interfaces*, vol. 35, no. 5, pp. 482–489, 2013.

[61] J. Mao and H. Cui, "Identifying bacterial biotope entities using sequence labeling: Performance and feature analysis," *Journal of the Association for Information Science and Technology*, vol. 69, no. 9, pp. 1134–1147, 2018.

[62] J. P. Chiu and E. Nichols, "Named entity recognition with bidirectional lstm-cnns," *Transactions of the association for computational linguistics*, vol. 4, pp. 357–370, 2016.

[63] X. Liu, Y. Zhou, and Z. Wang, "Recognition and extraction of named entities in online medical diagnosis data based on a deep neural network," *Journal of Visual Communication and Image Representation*, vol. 60, pp. 1–15, 2019.

[64] N. Che, D. Chen, and J. Le, "Entity recognition approach of clinical documents based on self-training framework," in *Recent Developments in Intelligent Computing, Communication and Devices*, Springer, 2019, pp. 259–265.

[65] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[66] C. Napoli, E. Tramontana, and G. Verga, "Extracting location names from unstructured italian texts using grammar rules and mapreduce," in *International Conference on Information and Software Technologies*, Springer, 2016, pp. 593–601.

[67] N. Pogrebnyakov, "Unsupervised domain-agnostic identification of product names in social media posts," in *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 3711–3716.

[68] S. Boytcheva, G. Angelova, Z. Angelov, and D. Tcharaktchiev, "Text mining and big data analytics for retrospective analysis of clinical texts from outpatient care," *Cybernetics and Information Technologies*, vol. 15, no. 4, pp. 58–77, 2015.

[69] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[70] D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao, "Relation classification via convolutional deep neural network," in *Proceedings of COLING 2014, the 25th international conference on computational linguistics: technical papers*, 2014, pp. 2335–2344.

[71] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[72] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext. zip: Compressing text classification models," *arXiv preprint arXiv: 1612.03651*, 2016.

[73] D. Zeng, K. Liu, Y. Chen, and J. Zhao, "Distant supervision for relation extraction via piecewise convolutional neural networks," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1753–1762.

[74] C. N. d. Santos, B. Xiang, and B. Zhou, "Classifying relations by ranking with convolutional neural networks," *arXiv preprint arXiv:1504.06580*, 2015.

[75] L. Wang, Z. Cao, G. De Melo, and Z. Liu, "Relation classification via multi-level attention cnns," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1298–1307.

[76] Z. He, W. Chen, Z. Li, W. Zhang, H. Shao, and M. Zhang, "Syntax-aware entity representations for neural relation extraction," *Artificial Intelligence*, vol. 275, pp. 602–617, 2019.

[77] S. Zhang, D. Zheng, X. Hu, and M. Yang, "Bidirectional long short-term memory networks for relation classification," in *Proceedings of the 29th Pacific Asia conference on language, information and computation*, 2015, pp. 73–78.

[78] M.-C. De Marneffe and C. D. Manning, "Stanford typed dependencies manual," Technical report, Stanford University, Tech. Rep., 2008.

[79] D. Sorokin and I. Gurevych, "Context-aware representations for knowledge base relation extraction," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1784–1789.

[80] D. Yang, S. Wang, and Z. Li, "Ensemble neural relation extraction with adaptive boosting," *arXiv preprint arXiv:1801.09334*, 2018.

[81] Y. Shen and X.-J. Huang, "Attention-based convolutional neural network for semantic relation extraction," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016, pp. 2526–2536.

[82] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, 2016, pp. 207–212.

[83] J. Du, J. Han, A. Way, and D. Wan, "Multi-level structured self-attentions for distantly supervised relation extraction," *arXiv preprint arXiv:1809.00699*, 2018.

[84] C. Alt, M. Hübner, and L. Hennig, "Fine-tuning pre-trained transformer language models to distantly supervised relation extraction," *arXiv preprint arXiv: 1906.08646*, 2019.

[85] H. Wang, M. Tan, M. Yu, S. Chang, D. Wang, K. Xu, X. Guo, and S. Potdar, "Extracting multiple-relations in one-pass with pre-trained transformers," *arXiv preprint arXiv:1902.01030*, 2019.

[86] G. Nan, Z. Guo, I. Sekulić, and W. Lu, "Reasoning with latent structure refinement for document-level relation extraction," *arXiv preprint arXiv:2005.06312*, 2020.

[87] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "Spanbert: Improving pre-training by representing and predicting spans," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, 2020.

[88] H. Wang, C. Focke, R. Sylvester, N. Mishra, and W. Wang, "Fine-tune bert for docred with two-step process," *arXiv preprint arXiv:1909.11898*, 2019.

[89] H. Tang, Y. Cao, Z. Zhang, J. Cao, F. Fang, S. Wang, and P. Yin, "Hin: Hierarchical inference network for document-level relation extraction," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2020, pp. 197–209.

[90] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[91] Y. Zhang, P. Qi, and C. D. Manning, "Graph convolution over pruned dependency trees improves relation extraction," *arXiv preprint arXiv:1809.10185*, 2018.

[92] Z. Guo, Y. Zhang, and W. Lu, "Attention guided graph convolutional networks for relation extraction," *arXiv preprint arXiv:1906.07510*, 2019.

[93] F. Christopoulou, M. Miwa, and S. Ananiadou, "Connecting the dots: Document-level neural relation extraction with edge-oriented graphs," *arXiv preprint arXiv:1909.00228*, 2019.

[94] L. Song, Y. Zhang, D. Gildea, M. Yu, Z. Wang, and J. Su, "Leveraging dependency forest for neural medical relation extraction," *arXiv preprint arXiv:1911. 04123*, 2019.

[95] T. Gao, X. Han, Z. Liu, and M. Sun, "Hybrid attention-based prototypical networks for noisy few-shot relation classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 6407–6414.

[96] L. B. Soares, N. FitzGerald, J. Ling, and T. Kwiatkowski, "Matching the blanks: Distributional similarity for relation learning," *arXiv preprint arXiv: 1906.03158*, 2019.

[97] T. Gao, X. Han, R. Xie, Z. Liu, F. Lin, L. Lin, and M. Sun, "Neural snowball for few-shot relation learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 7772–7779.

[98] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 2009, pp. 1003–1011.

[99] Y. Lin, Z. Liu, and M. Sun, "Neural relation extraction with multi-lingual attention," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 34–43.

[100] G. Ji, K. Liu, S. He, and J. Zhao, "Distant supervision for relation extraction with sentence-level attention and entity descriptions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[101] X. Feng, J. Guo, B. Qin, T. Liu, and Y. Liu, "Effective deep memory networks for distant supervised relation extraction.," in *IJCAI*, vol. 17, 2017.

[102] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[103] J. Feng, M. Huang, L. Zhao, Y. Yang, and X. Zhu, "Reinforcement learning for relation classification from noisy data," in *Proceedings of the aaai conference on artificial intelligence*, vol. 32, 2018.

[104] P. Qin, W. Xu, and W. Y. Wang, "Robust distant supervision relation extraction via deep reinforcement learning," *arXiv preprint arXiv:1805.09927*, 2018.

[105] K. Yang, L. He, X. Dai, S. Huang, and J. Chen, "Exploiting noisy data in distant supervision relation classification," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 3216–3225.

[106] X. Zeng, S. He, K. Liu, and J. Zhao, "Large scaled relation extraction with reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[107] M. Miwa and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures," *arXiv preprint arXiv:1601.00770*, 2016.

[108] A. Katiyar and C. Cardie, "Investigating lstms for joint extraction of opinion entities and relations," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 919–929.

[109] S. Zheng, F. Wang, H. Bao, Y. Hao, P. Zhou, and B. Xu, "Joint extraction of entities and relations based on a novel tagging scheme," *arXiv preprint arXiv:1706.05075*, 2017.

[110] X. Zeng, D. Zeng, S. He, K. Liu, and J. Zhao, "Extracting relational facts by an end-to-end neural model with copy mechanism," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 506–514.

[111] D. Zeng, H. Zhang, and Q. Liu, "Copymtl: Copy mechanism for joint extraction of entities and relations with multi-task learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 9507–9514.

[112] T.-J. Fu, P.-H. Li, and W.-Y. Ma, "Graphrel: Modeling text as relational graphs for joint entity and relation extraction," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 1409–1418.

[113] B. Distiawan, G. Weikum, J. Qi, and R. Zhang, "Neural relation extraction for knowledge base enrichment," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 229–240.

[114] T. Nayak and H. T. Ng, "Effective modeling of encoder-decoder architecture for joint entity and relation extraction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 8528–8535.

[115] B. Yu, Z. Zhang, X. Shu, Y. Wang, T. Liu, B. Wang, and S. Li, "Joint extraction of entities and relations based on a novel decomposition strategy," *arXiv preprint arXiv:1909.04273*, 2019.

[116] Y. Yuan, X. Zhou, S. Pan, Q. Zhu, Z. Song, and L. Guo, "A relation-specific attention network for joint entity and relation extraction.," in *IJCAI*, vol. 2020, 2020, pp. 4054–4060.

[117] R. Takanobu, T. Zhang, J. Liu, and M. Huang, "A hierarchical framework for relation extraction with reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 7072–7079.

[118] D. Sui, Y. Chen, K. Liu, J. Zhao, X. Zeng, and S. Liu, "Joint entity and relation extraction with set prediction networks," *arXiv preprint arXiv:2011.01675*, 2020.

[119] J. Wang and W. Lu, "Two are better than one: Joint entity and relation extraction with table-sequence encoders," *arXiv preprint arXiv:2010.03851*, 2020.

[120] K. Sun, R. Zhang, S. Mensah, Y. Mao, and X. Liu, "Recurrent interaction network for jointly extracting entities and classifying relations," *arXiv preprint arXiv:2005.00162*, 2020.

[121] B. Ji, J. Yu, S. Li, J. Ma, Q. Wu, Y. Tan, and H. Liu, "Span-based joint entity and relation extraction with attention-based span-specific and contextual semantic representations," in *Proceedings of the 28th International Conference on Computational Linguistics*, 2020, pp. 88–99.

[122] Z. Wei, J. Su, Y. Wang, Y. Tian, and Y. Chang, "A novel cascade binary tagging framework for relational triple extraction," *arXiv preprint arXiv:1909.03227*, 2019.

[123] Y. Wang, B. Yu, Y. Zhang, T. Liu, H. Zhu, and L. Sun, "Tplinker: Single-stage joint extraction of entities and relations through token pair linking," *arXiv preprint arXiv:2010.13415*, 2020.

[124] H. Ye, N. Zhang, S. Deng, M. Chen, C. Tan, F. Huang, and H. Chen, "Contrastive triple extraction with generative transformer," *arXiv preprint arXiv:2009.06207*, 2020.

[125] M. Surdeanu, J. Tibshirani, R. Nallapati, and C. D. Manning, "Multi-instance multi-label learning for relation extraction," in *Proceedings of the 2012 joint Conf. on empirical methods in natural language processing and computational natural language learning*, 2012, pp. 455–465.

[126] S. Riedel, L. Yao, and A. McCallum, "Modeling relations and their mentions without labeled text," in *Joint European Conf. on Machine Learning and Knowledge Discovery in Databases*, Springer, 2010, pp. 148–163.

[127] J. Ebrahimi and D. Dou, "Chain based rnn for relation classification," in *Proceedings of the 2015 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 1244–1249.

[128] K. Sun, R. Zhang, Y. Mao, S. Mensah, and X. Liu, "Relation extraction with convolutional network over learnable syntax-transport graph.," in *AAAI*, 2020, pp. 8928–8935.

[129] A. Pouran Ben Veyseh, T. Nguyen, and D. Dou, "Improving cross-domain performance for relation extraction via dependency prediction and information flow control," in *Proc. of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, Jul. 2019, pp. 5153–5159.

[130] Y. Liu, K. Liu, L. Xu, and J. Zhao, "Exploring fine-grained entity type constraints for distantly supervised relation extraction," in *Proceedings of COLING 2014, the 25th Int. Conf. on Computational Linguistics*, 2014, pp. 2107–2116.

[131] P. Xu and D. Barbosa, "Connecting language and knowledge with heterogeneous representations for neural relation extraction," in *Proceedings of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, Minneapolis, Minnesota, Jun. 2019, pp. 3201–3206.

[132] J. Nivre, M.-C. de Marneffe, F. Ginter, J. Hajič, C. D. Manning, S. Pyysalo, S. Schuster, F. Tyers, and D. Zeman, "Universal Dependencies v2: An evergrowing multilingual treebank collection," in *Proceedings of the 12th Language Resources and Evaluation Conference*, May 2020, pp. 4034–4043.

[133] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, "Stanza: A python natural language processing toolkit for many human languages," *arXiv preprint arXiv:2003.07082*, 2020.

[134] Y. Lin, S. Shen, Z. Liu, H. Luan, and M. Sun, "Neural relation extraction with selective attention over instances," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016, pp. 2124–2133.

[135] G. Zhou, J. Su, J. Zhang, and M. Zhang, "Exploring various knowledge in relation extraction," in *Proceedings of the 43rd annual meeting of the association for computational linguistics (aclâ05)*, 2005, pp. 427–434.

[136] Y. S. Chan and D. Roth, "Exploiting syntactico-semantic structures for relation extraction," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011, pp. 551–560.

[137] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," *arXiv preprint arXiv:1603.01360*, 2016.

[138] X. Ren, Z. Wu, W. He, M. Qu, C. R. Voss, H. Ji, T. F. Abdelzaher, and J. Han, "Cotype: Joint extraction of typed entities and relations with knowledge bases," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 1015–1024.

[139] M. Miwa and Y. Sasaki, "Modeling joint entity and relation extraction with table representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1858–1869.

[140] A. Katiyar and C. Cardie, "Going out on a limb: Joint extraction of entity mentions and relations without dependency trees," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 917–928.

[141] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang, *et al.*, "Abstractive text summarization using sequence-to-sequence rnns and beyond," *arXiv preprint arXiv:1602.06023*, 2016.

[142] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[143] S. Rothe, S. Narayan, and A. Severyn, "Leveraging pre-trained checkpoints for sequence generation tasks," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 264–280, 2020.

[144] D. Zelenko, C. Aone, and A. Richardella, "Kernel methods for relation extraction," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1083–1106, 2003.

[145] M. R. Gormley, M. Yu, and M. Dredze, "Improved relation extraction with feature-rich compositional embedding models," *arXiv preprint arXiv:1505.02419*, 2015.

[146] X. Yu and W. Lam, "Jointly identifying entities and extracting relations in encyclopedia text via a graphical model approach," in *Coling 2010: Posters*, 2010, pp. 1399–1407.

[147] J. Chen, C. Yuan, X. Wang, and Z. Bai, "Mrmep: Joint extraction of multiple relations and multiple entity pairs based on triplet attention," in *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, 2019, pp. 593–602.

[148] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H.-W. Hon, "Unified language model pre-training for natural language understanding and generation," *arXiv preprint arXiv:1905.03197*, 2019.

[149] Y.-C. Chen, Z. Gan, Y. Cheng, J. Liu, and J. Liu, "Distilling knowledge learned in bert for text generation," *arXiv preprint arXiv:1911.03829*, 2019.

[150] A. Wang and K. Cho, "Bert has a mouth, and it must speak: Bert as a markov random field language model," *arXiv preprint arXiv:1902.04094*, 2019.

[151] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, "Mass: Masked sequence to sequence pre-training for language generation," *arXiv preprint arXiv:1905.02450*, 2019.

[152] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[153] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[154] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, "Creating training corpora for nlg micro-planning," in *55th annual meeting of the Association for Computational Linguistics (ACL)*, 2017.

[155] Q. Guo, F. Zhuang, C. Qin, H. Zhu, X. Xie, H. Xiong, and Q. He, "A survey on knowledge graph-based recommender systems," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[156] J. Chicaiza and P. Valdiviezo-Diaz, "A comprehensive survey of knowledge graph-based recommender systems: Technologies, development, and contributions," *Information*, vol. 12, no. 232, pp. 9507–9514, 2021.

[157] M. Yasunaga, H. Ren, A. Bosselut, P. Liang, and J. Leskovec, *Qa-gnn: Reasoning with language models and knowledge graphs for question answering*, 2021. arXiv: 2104.06378 [cs.CL].

[158] L. Liu, B. Du, H. Ji, and H. Tong, "Kompare: A knowledge graph comparative reasoning system," in *Proceedings of KDD'21*, Online, Singapore, 2021, 3308?3318.

[159] A. Chan, S. Sanyal, B. Long, J. Xu, T. Gupta, and X. Ren, "Salkg: Learning from knowledge graph explanations for commonsense reasoning," in *NeurIPS 20211*, Online, Sydney, Australia, 2021.

[160] F. Liu, C. You, X. Wu, S. Ge, and S. W. anb Xu Sun, "Auto-encoding knowledge graph for unsupervised medical report generation," in *NeurIPS 20211*, Online, Sydney, Australia, 2021.

[161] Z. Zhang, J. Wang, J. Chen, S. Ji, and F. Wu, "Cone: Cone embeddings for multi-hop reasoning over knowledge graphs," in *NeurIPS 20211*, Online, Sydney, Australia, 2021
.