Knowledge Extraction from Unstructured Text

by
Madhusudan Ghosh

Exam Roll No.-M4CSE19004

Registration NO.-140757 0f 2017-2018

Class Roll No.-001710502018

Session:2017-2019

This dissertation is submitted for the degree of Master of Engineering

Under the Guidance and Supervision of
Dr. Sudip Kumar Naskar
Jadavpur University
Kolkata-700032

Declaration Of Authorship

I, hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of Master in Computer Science and Engineering studies.
All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.
I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials that are not original to this work.

Sign: _____ **Date:** _____
Name: Madhusudan Ghosh
Examination Roll No. M4CSE19004
Registration No. 140757 0f 2017-2018
Thesis Title: Knowledge Extraction from Unstructured Text.

Certificate of Recommendation

This is to certify that the dissertation entitled "Knowledge Extraction from Unstructured Text" has been carried out by Madhusudan Ghosh (University Registration No.: 140757 of 2017-18, Examination Roll No.: M4CSE18012) under my guidance and supervision and be accepted in partial fulfillment of the requirement for the Degree of Master in Computer Science and Engineering. The research results presented in the thesis have not been included in any other thesis submitted for the award of any degree in any other University or Institute.

Dr. Sudip Kumar Naskar
Dept. Of Computer Science Engineering
Jadavpur University, Kolkata-700032

Signature: _____

Prof. Mahantapas Kundu
Head Of Department
Dept. Of Computer Science and Engineering
Jadavpur University, Kolkata-700032

Signature: _____

Prof. Chiranjib Bhattacharjee
Dean
Faculty of Engineering and Technology
Jadavpur University, Kolkata-700032

Signature: _____

Certificate Of Approval

This is to certify that the thesis entitled "Knowledge Extraction from Unstructured Text" is a bonafide record of work carried out by Madhusudan Ghosh **(University Registration No: 140757 of 2017-18, Examination Roll No:M4CSE19004)** in partial fulfillment of the requirements for the award of the degree of Master in Computer Science and Engineering in the Department of Computer Science and Engineering, Jadavpur University, during the period of June 2018 to May 2019. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, the opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

Examiner:

Signature: _____ **Date:** _____

Dr. Sudip Kr. Naskar
Dept. Of Computer Science and Engineering
Jadavpur University, Kolkata-700032

Signature: _____ **Date:** _____

Acknowledgement

I would like to express my sincere gratitude to my advisor, **Dr. Sudip Kumar Naskar**, for his continuous motivation, guidance and patience throughout my thesis work. I have been very lucky to have an advisor who cared so much about my work. More importantly, the scientific and personal support along with his valuable suggestions softened the journey.
I am very much grateful to **Subhabrata Dutta** for his continuous help and valuable suggestions. I am also thankful to all other NLP lab members for their continuous support.

Madhusudan Ghosh
Signature: _____

Abstract

Knowledge Graph plays a important role in many tasks in Natural language processing (Question Answering system, Summarization, etc.). Extracting Knowledge from unstructured text is a very challenging research problem. This problem becomes particularly acute due to ambiguities and lexical variations in Natural Language. Most of the knowledge available to the humankind are in the form of unstructured text. However, these knowledge is not directly accessible to machines unless they are converted into structured format. The purpose of this research is to extract useful pieces of information (or knowledge) in the form of RDF (S-O-P triplet) from unstructured textual contents which will be useful for populating Knowledge Bases. In this thesis, we explored different deep learning based architectures (LSTM, Bi-LSTM, GCN) for the task of knowledge extraction from unstructured text.

Contents

Chapter 1

Introduction

1.1 Background

With the arrival of the information age, world knowledge is accumulating at an astronomical rate. Since on-line information is stored in the form of unstructured documents that are hard to classify and search on, without making the information efficiently accessible, the ratio of attainable knowledge over existing information will be extremely low. One example of on-line information growing unmanageable is electronic paper. If one spends 30 minutes to read a paper, it will take a huge amount of time. If one only reads the abstracts which takes five minutes each, it still requires quite some time to complete. But if information is available in the form of summary for such papers, it will be much more beneficial to the readers.

However, after reading huge amount of sentences, can a human being remember how many times a knowledge has been repeated or if there is any knowledge conflict either directly or indirectly with the knowledge already encountered? Not only retrieving knowledge from the vast source of information is needed for efficient knowledge management, but also the ability to validate the knowledge extracted against the current knowledge base has become crucial in achieving a consistent and reliable knowledge base. This situation motivates the research of this thesis. The goal of this thesis work is to develop a methodology that can take any unstructured text corpus, extract the knowledge present in it and turn them into a format of RDF (Subject-Object-Predicate) which will be useful to populate the Knowledge Base.

1.2 Motivation

The world wide web is a vast repository of knowledge, with data present in multiple modalities such as text, videos, images, structured tables, etc. However, most of the data is present in unstructured format and extracting

information(i.e. Knowledge) in structured machine-readable format is still a very difficult task. Knowledge graphs aim at constructing large repositories of structured knowledge which can be easily processed by machines. Such knowledge graphs are being used to improve the relevance and the quality of search in case of search engines like Google and Bing. Knowledge graphs are also being used by applications like Google now, Microsoft Cortana and Apple Siri which are capable of understanding natural language queries and answer questions, making recommendations, etc. to the user. The construction of knowledge graphs is thus a major step towards making intelligent personalized machines.

1.3 Different Types of Approaches for Knowledge Extraction System

There are three types of Knowledge Extraction System. All these approaches are discussed in the following subsections.

1.3.1 Supervised approaches

Supervised models used in the field of information extraction involves formulation of the problem as a classification problem and they generally learn a discriminative classifier given a set of positive and negative examples. Such approaches extract a set of features from the sentence, which generally include context words, part of speech (POS) tags, dependency path between entity pairs, edit distance, etc. and the corresponding labels are obtained from a large labelled training corpus.
Although such methods obtain good accuracies and take into account negative examples explicitly for relation extraction, they are neither general nor scalable. Such methods are very expensive due to requirement of large amount of training data. Moreover, the relations learned from these methods are largely dependent on the domain and thus cannot be generalized.

1.3.2 Semi supervised approaches

Semi supervised approaches have been used for a long time in the field of relation extraction from unstructured text. Bootstrapping methods for relation extraction fall in this category. Such methods start with some known relation triples and then iterate through the text to extract patterns that match the seed triples. These patterns are used to extract more relations from the data set and the learned relations are then added to the seed examples. This method is repeated till no more relations can be learned from the data set. Some of the most popular approaches such as Dual Iterative Pattern Snowball, Text Runner are examples of semi supervised methods. These projects, however, rely heavily on the correctness of NLP tools (e.g.,

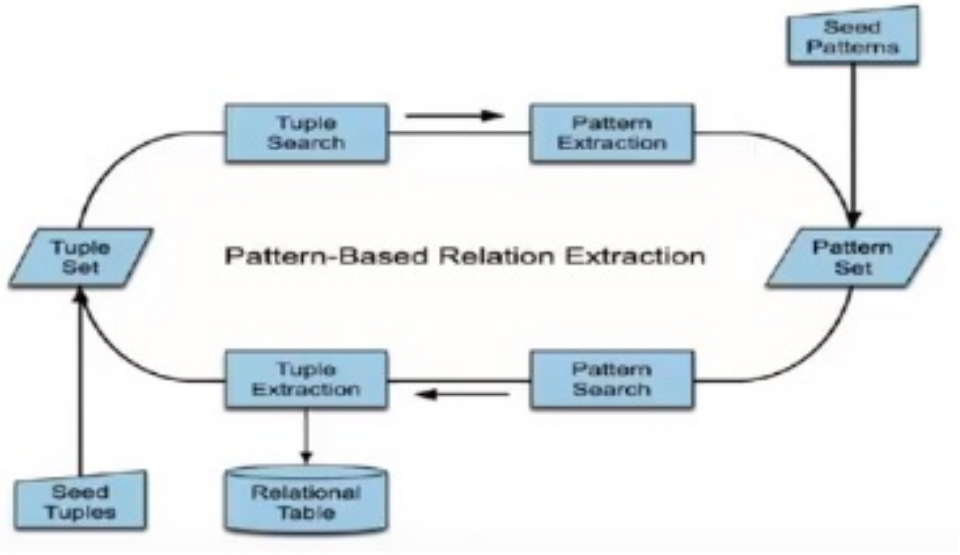


Figure 1.1: Semi-supervised Approach

Named Entity Recognition (NER)) and thus they may be prone to errors. For example, TextRunner extracts relations automatically from the text corpus using NLP tools like dependency parser and Noun Phrase chunker. Semi supervised approaches can be used to learn relations of a particular type accurately. However, seed examples are needed for that particular relation type and thus may require larger supervision for learning general knowledge graphs from different domains. Semi supervised relation extraction approach is depicted in Figure 1.1.

1.3.3 Distant supervision

In case of distant supervision methods for relation extraction, existing knowledge bases are used with large text corpus to generate a large number of relation triples. Such relations are located in the text and hypotheses are learnt corresponding to these examples. These hypotheses can be combined to learn a generalized model for relation extraction.
Projects, e.g. NELL, use distant supervision methods for learning relations. They use predefined ontology and then bootstrap relations from the web and text using positive and negative seed examples of ontology constraints. Later, they use multi-view learning paradigm to extract entity relations from unstructured text and the web. Use of multiple hypotheses and an objective function based on agreement and disagreement of these hypotheses ensures less noise while expansion. Often these hypotheses are hand-crafted and coming up with them require domain expertise.

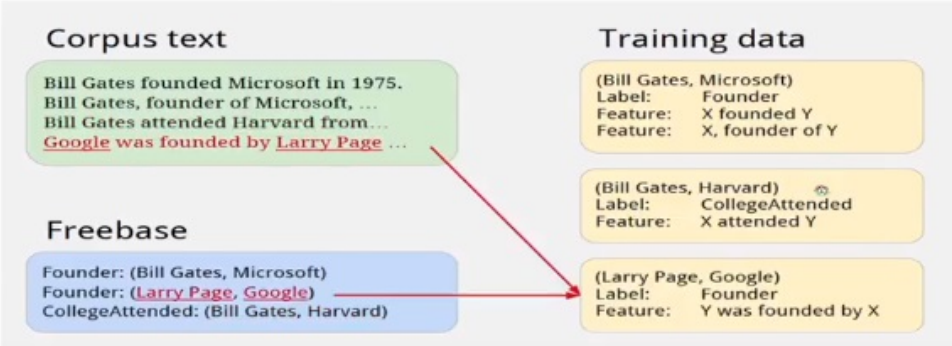


Figure 1.2: An example of Distant Supervision Approach

These methods have distinct advantages over other methods since they are scalable and require almost no supervision. These models can also be used to learn relations from general domains. However, such methods are computationally expensive and may learn wrong relation-expression hypotheses which can generate a large number of false relations. An example of Distant Supervision approach is mentioned in 1.2.
Most of the above stated methods for relation extraction utilizes syntactic information either in the training phase or in the evaluation. Hence, these methods can only be used to learn relations from English corpus and fails for other languages like Hindi, for which there are no efficient and reliable syntactic NLP tools. This motivated us to explore distributed word vector representations for learning relations which does not rely on syntactic information and extract semantic meaning of words based on their context. Thus, there is a scope to extend these methods to extract relations from corpus in any language.

Chapter 2

Overview of different Knowledge Bases

In this chapter, we will discuss about different knowledge bases, with a specfic focus on Wikidata, and their architectures.

2.1 Wikidata

Wikidata is a collaboratively edited knowledge base hosted by the Wikimedia Foundation. It is a common source of open data under Wikimedia which can be used by Wikipedia and anyone else, under a public domain license. In a similar way, Wikimedia Commons provides storage for media files and access to those files for all Wikimedia projects, which are also freely available for reuse. Wikidata is powered by the software Wikibase.
Wikibase is a set of MediaWiki extensions for working with versioned data in a central repository. Its primary components are the Wikibase Repository, an extension for storing and managing data, and the Wikibase Client which allows for the retrieval and embedding of structured data from a wikibase repository.

2.1.1 Data Model

The data model of **WikibaseLexeme** describes the structure of the data that is handled as "Lexeme" in Wikibase.
A Lexeme is a lexical element of a language, such as a word, a phrase, or a prefix. Lexemes are Entities in the sense of the Wikibase data model. A Lexeme is described using the following information:

- Lexemes have IDs starting with a "L" followed by a natural number in decimal notation, e.g. L3746552. These IDs are unique within the repository that manages the Lexeme. The ID can be combined with a repository's concept base URI to form a unique URI for the Lexeme.

- A Lemma is a readable representation of the lexeme, e.g. "run".
- The Language to which the **lexeme** belongs to. This is a reference to a concrete Item, e.g. English (Q1860).
- A Lexeme Statement is described by the properties of the lexeme that are not specific to a Form or Sense (e.g. derived from or grammatical gender or syntactic function).
- A list of Forms, typically one for each relevant combination of grammatical features, such as 2nd person or singular or past tense. A Form is described using the following information:
  - Forms have IDs starting with the ID of the Lexeme they belong to, followed by a hyphen ("-") and an "F", followed by a natural number in decimal notation: e.g. L3746552-F7.
  - A representation, spelling out the Form as a string.
  - A list of grammatical features for which syntactic roles are applied. These are given as references to a concrete Items.
  - A list of Form Statements further describing the Form or its relations to other Forms or Items (e.g. IPA transcription (P898), pronunciation audio, rhymes with, used until, used in region)
- A list of **Senses**, describing the different meanings of the lexeme (e.g. "financial institution" and "edge of a body of water" for the English noun bank). A sense is described using the following information:
  - Senses have IDs starting with the ID of the Lexeme they belong to, followed by a hyphen ("-") and an "S", followed by a natural number in decimal notation: e.g. L3746552-S4. These IDs are unique within the repository that manages the Lexeme. The ID can be combined with a repository's concept base URI to form a unique URI for the Sense.
  - A Gloss, defining the meaning of the Sense using natural language.
  - A list of Sense Statements further describing the Sense and its relations to Senses and Items (e.g. translation, synonym, antonym, connotation, register, denotes, evokes).

The above one is the concrete structure for Wikidata model if we can see it in pictorial referred to diagram 2.1.

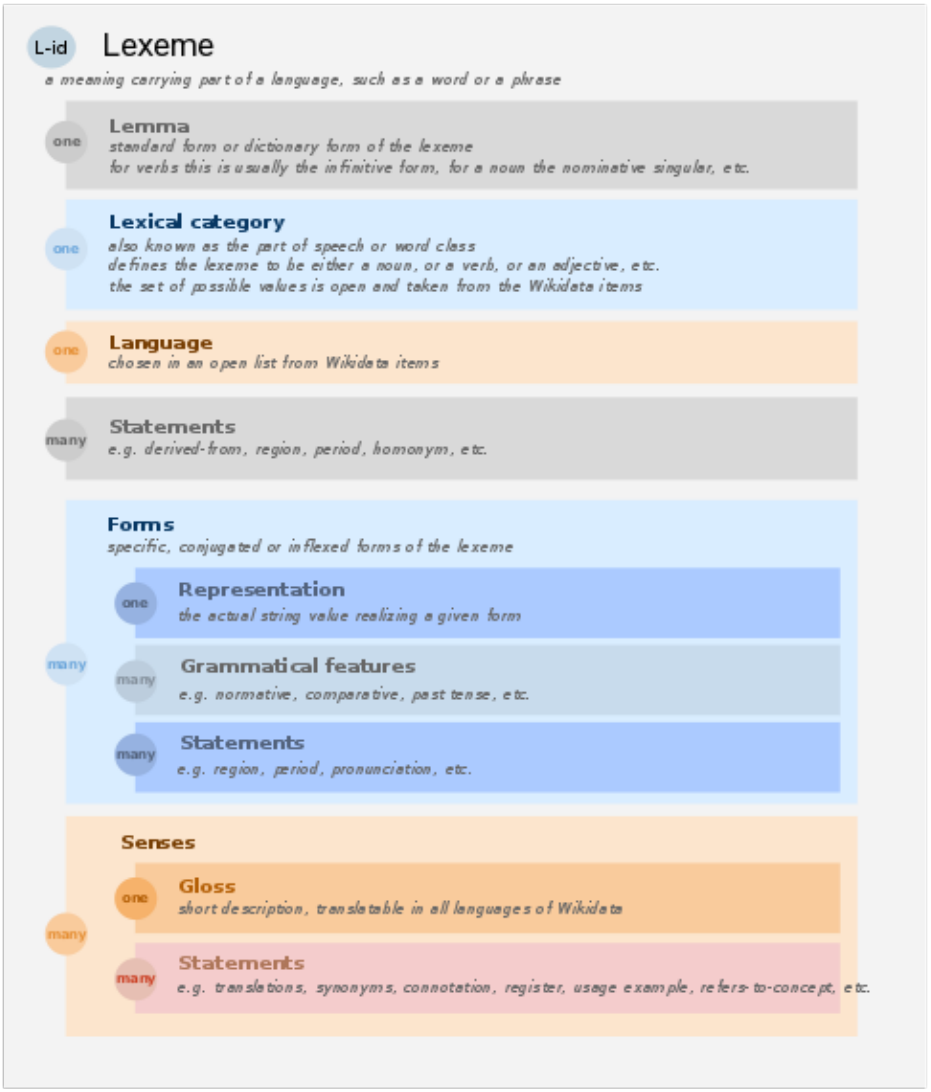


Figure 2.1: Architecture of Wikidata

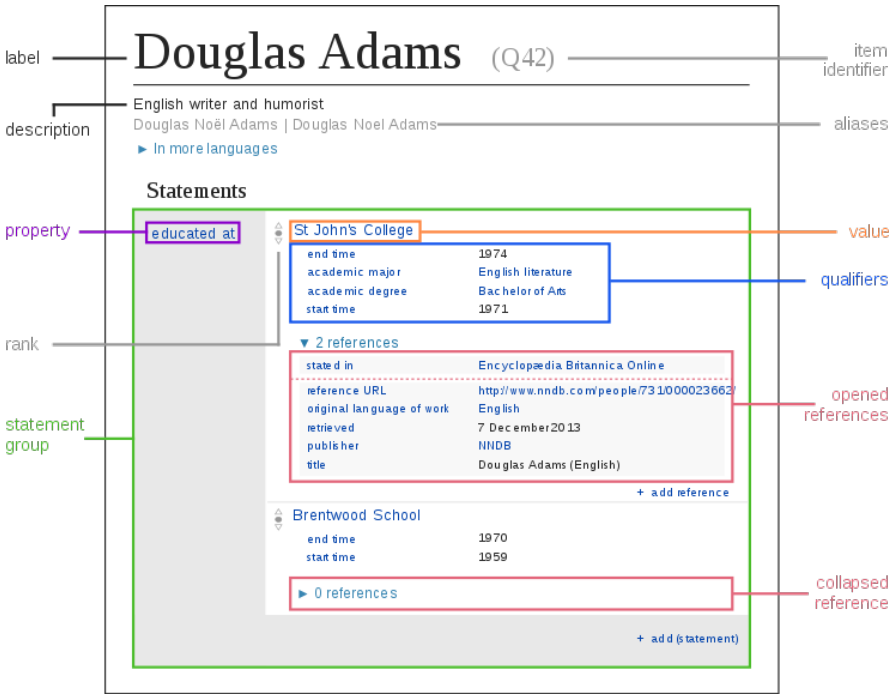


Figure 2.2: Wikidata Representation for a particular Person

2.1.2 Working Process of Wikidata

Wikidata is a central storage repository that can be accessed by others, such as by the Wikimedia Foundation. Content loaded dynamically from Wikidata does not need to be maintained in each individual wiki project. For example, statistics, dates, locations and other common data can be centralized in Wikidata.

Wikidata Repository

The Wikidata repository consists mainly of items, each one having a label, a description and any number of aliases. Items are uniquely identified by a Q followed by a number, such as Douglas Adams (Q42).

Statements describe detailed characteristics of an Item and consist of a property and a value. Properties in Wikidata have a P followed by a number, such as with educated at (P69).
For a person, we can add a property to specify where they were educated, by specifying a value for a school. For buildings, we can assign geographic coordinates properties by specifying longitude and latitude values. Properties can also link to external databases. A property that links an item to an external database, such as an authority control database used by li-

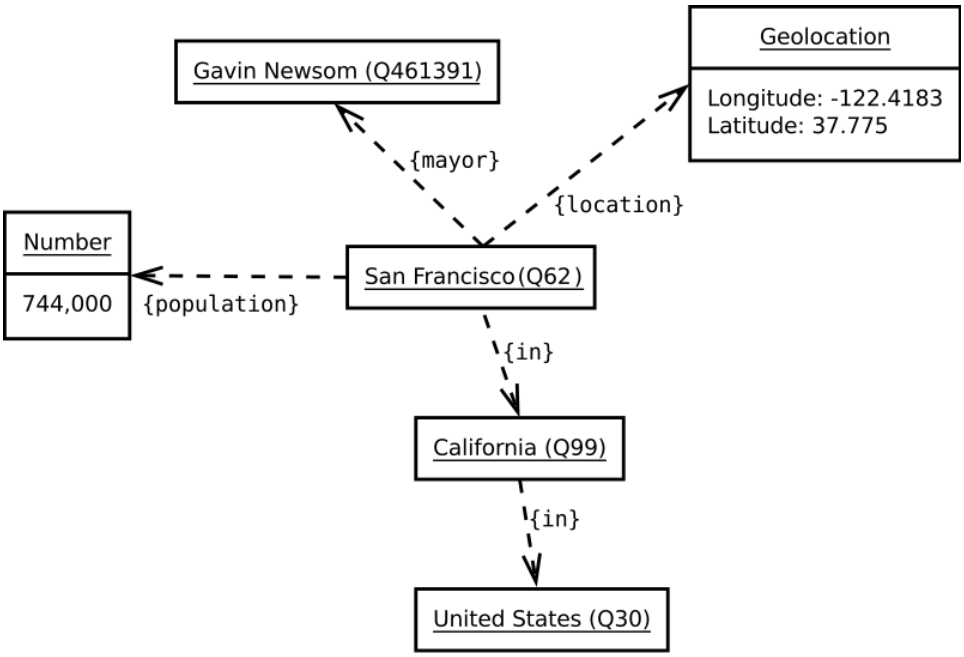


Figure 2.3: Graphical Representation of any Wikidata Entity

braries and archives, is called an **identifier**. Special Sitelinks connect an item to corresponding content on client wikis, such as Wikipedia, Wikibooks or Wikiquote.We can see the above concept in a diagram 2.2.

The knowledge graph can be depicted by the following way and it can be represented in the diagram2.3. All this information can be displayed in any language, even if the data originated in a different language. When accessing these values, client wikis will show the most up-to-date data.

Item	Property	Value
Q42	P69	Q691283
Douglas Adams	educated at	St John's College

Wikidata is an ongoing project that is under active development. More data types as well as extensions will be available in the future. We can find more information about Wikidata and its ongoing development on the Wikidata page on Meta.

Like Wikidata there are many more existing knowledge bases we am just going to give some basic overview about some of them.

2.2 ConceptNet

ConceptNet is a semantic network based on the information in the OMCS(Open Mind Common Sense)database. ConceptNet is expressed as a directed graph whose nodes are concepts, and whose edges are assertions of common sense about these concepts.

Concepts represent sets of closely related natural language phrases, which could be noun phrases, verb phrases, adjective phrases, or clauses.

ConceptNet is created from the natural-language assertions in OMCS by matching them against patterns using a shallow parser. Assertions are expressed as relations between two concepts, selected from a limited set of possible relations. The various relations represent common sentence patterns found in the OMCS corpus, and in particular, every "fill-in-the-blanks" template used on the knowledge-collection Web site is associated with a particular relation.

ConceptNet is a knowledge representation project, providing a large semantic graph that describes general human knowledge and how it is expressed in natural language. The scope of ConceptNet includes words and common phrases in any written human language. It provides a large set of background knowledge that a computer application working with natural language text should know. These words and phrases are related through an open domain of predicates, describing not just how words are related by their lexical definitions, but also how they are related through common knowledge. A cluster of related concepts and the ConceptNet assertions that connect them is visualized in Figure 2.4.

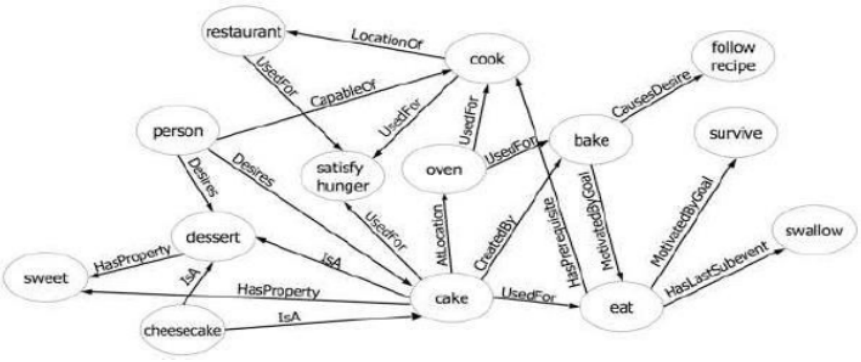


Figure 2.4: ConceptNet Representation

The new goals of ConceptNet 5 is to include knowledge from other crowdsourced knowledge with their own communities and editing processes, particularly data mined from Wiktionary and Wikipedia; to add links to other resources such as DBPedia, Freebase, and WordNet to support machine-reading tools such as ReVerb , which extracts relational knowledge from

Web pages and to find translations between concepts represented in different natural languages.

2.3 WordNet

WordNet is a lexical database for the English language. It groups English words into sets of synonyms called synsets, provides short definitions and usage examples, and records a number of relations among these synonym sets or their members. WordNet can thus be seen as a combination of dictionary and thesaurus. While it is accessible to human users via a web browser, its primary use is in automatic text analysis and artificial intelligence applications. The database and software tools have been released under a BSD style license and are freely available for download from the WordNet website. Both the lexicographic data (lexicographer files) and the compiler (called grind) for producing the distributed database are available.

2.4 BabelNet

BabelNet is a multilingual lexicalized semantic network and ontology developed at the Sapienza University of Rome, at the Department of Computer Science Linguistic Computing Laboratory, BabelNet was automatically created by linking Wikipedia to the most popular computational lexicon of the English language, WordNet . The integration is done using an automatic mapping and by filling in lexical gaps in resource-poor languages by using statistical machine translation. The result is an "encyclopedic dictionary" that provides concepts and named entities lexicalized in many languages and connected with large amounts of semantic relations. Additional lexicalizations and definitions are added by linking to free-license wordnets, OmegaWiki, the English Wiktionary, Wikidata, FrameNet, VerbNet and others. Similarly to WordNet, BabelNet groups words in different languages into sets of synonyms, called Babel synsets. For each Babel synset, BabelNet provides short definitions (called glosses) in many languages harvested from both WordNet and Wikipedia.

2.5 DBpedia

From the paper [11] we can get a thorough overview about this KnowledgeBase.The DBpedia community project extracts structured, multilingual knowledge from Wikipedia and makes it freely available using Semantic Web and Linked Data standards. The extracted knowledge, comprising more than 1.8 billion facts, is structured according to an ontology maintained by the community. The knowledge is obtained from different Wikipedia language editions, thus covering more than 100 languages, and mapped to

the community ontology. The resulting data sets are linked to more than 30 other data sets in the Linked Open Data (LOD) cloud. The DBpedia project was started in 2006 and has meanwhile attracted large interest in research and practice. Being a central part of the LOD cloud, it serves as a connection hub for other data sets. For the research community, DBpedia provides a tested serving real world data spanning many domains and languages. Due to the continuous growth of Wikipedia, DBpedia also provides an increasing added value for data acquisition, re-use and integration tasks within organisations. Here, we give an overview over the DBpedia community project, including its architecture, technical implementation, maintenance, internationalisation, usage statistics and showcase some popular DBpedia applications.

Chapter 3

Related Work

In this chapter we will discuss all the related works which were already done on Knowledge Extraction research field.

3.1 Knowledge Extraction using Rule Based Approach

From this paper[7] We come to know that the authors used novel approach combination of Rule Based approach and also the Similarity based approach.The architecture that was proposed by the authors is depicted in diagram 3.1.

In the above diagram the Unstructured text was given as input then the authors did some preprocessing tasks which were essential for the Knowledge Extraction task in the following way.

3.1.1 Entity Mapping

Extracted entities from the text was mapped to Knowledge Base entities by giving KB's URI (i.e.Uniform Resource Identifier) if any entity was not found in the Knowledge Base then a random Uniform Resource Identifier was given to that particular entity or entities.

3.1.2 Coreference Resolution

Then the text was pushed to the Coreference Resolver(Stanford Corenlp Resolver) to get the Co-referring entities.

3.1.3 Triple Extraction

This is the main part of extracting knowledge from the unstructured text. To extract a entity-relation triple an open information technique was used which is template based approach.

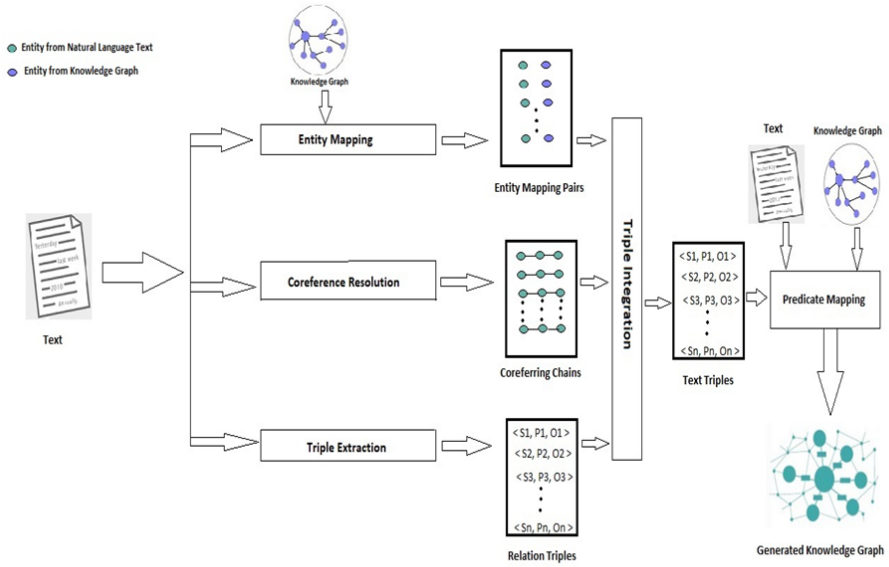


Figure 3.1: Architecture of Relation Extraction using Rule-based Approach

3.1.4 Triple Integration

In the Triple Extraction component, anyone can extract relation triples from unstructured text; however, entity mapping and coreference resolution among the entities of such triples were not performed.As a result, ambiguity in the triple occurred and interlinking to entities in the KB was not established. Consequently, transformation of a relation triple that conformed to the standard of KB was required. Therefore, to deal with such problems, the authors integrated and transformed three components by the following process.

First, identical entities were grouped by using coreferring chains from the Coreference Resolution component. Second, a representative for the group of coreferring entities was selected using the voting algorithm. Third, all entities belonging to the group in the relation triples were replaced by the representative of its group. Then, the relation of a relation triple was straightforwardly transformed into a predicate by assigning a new URI. Finally, if an object of a relation triple was not an entity, it was left as literal. After performing these processes, text triples were extracted from unstructured text.

3.1.5 Predicate Mapping

The main aim of this portion is to map a predicate of a triple to its corresponding predicate in KB. For the above task the authors used rule based

approach by using the enriched triples which were generated using the extracted text triples and also the KB base triples. The output of the above task was used for generating bootstrapping triples which will be useful for the next module. Then the authors applied one approach as follows :

Rule-Based Candidate Generation

The rules for generating candidates were as follows:

If the subject and object of the text triple was same as the subject and the object of the KB triple then it was assumed that the predicate of the text triple was same as the predicate of KB triple.

The class of the subject and the class of the object were used as a constraint for mapping. For example, a finding rule can be (**Person, ex: born in, Location**) mapped to **dbpedia : birthPlace (using DBpedia as the KB)**.

Candidate Generation

The Candidate Selection module selected the mapping for the predicate of the text triple. Here, priority was given to the predicate candidate pair, which was generated by the Rule-based Candidate Generation module. The output of candidate selection was the generated KB, in which both the entities and the predicates were linked to other KBs.

3.2 Feature Based Methods

From the [6] we came to know that a set of relevant features were designed by domain experts for a classification problem. Later this set of features were given to classifier for training and classification purpose. For relation extraction task, sentences with predefined entities were used to construct feature vector through feature extraction process. Commonly used feature for relation extraction task are described below.

3.2.1 Lexical Features

In this feature set, lexical features such as position of mentioned pair of entities, number of words between mentioned pair, etc. were used to capture context of the text and bag of word model also helped to represent sentence and words as a feature in our feature vector[20].

3.2.2 Syntax Tree Features:

In this feature set, grammatical structure of the sentence and mentioned pair were used for feature creation. For example, part of speech tags for

each mentioned pair, chunk head, etc., were used as a feature for relation extraction.

3.2.3 Entity Features

A relation could exist between certain type of entities, for example Treatment. Medical Problem could exist between a treatment entity and problem entity.So, type of mentioned pair of entities were also important feature values for classification purpose.

3.3 Distant Supervision Approach

This approach was proposed by the paper[17]. It is semi-supervised approach for generating relation triples from the large text corpus using small amount of labelled data.Throughout the experiment Freebase was used as Knowledge Base.

Authors proposed a very much novel approach which is being used in a large scale recent times for data generation. The idea was that if a sentence contained two entities and those two entities were an instance of one of any Freebase relation, then features were extracted and added to the feature vector.In training time, the features for identical tuples (relation, entity1, entity2) from different sentences were combined and a richer feature vector was created. During the testing time entities were again identified using the named entity tagger. This time, every pair of entities appearing together in a sentence was considered a potential relation instance, and whenever those entities appear together, features were extracted on the sentence and added to a feature vector for that entity pair. Authors used logistic regression as a relation classifier for that they used the following features.

3.3.1 Features

Each features described how two entities were related to the system using syntactic or non-syntactic information.

3.3.2 Lexical Feature:

The lexical features were as follows:

- The sequence of words between the two entities.
- The part-of-speech tags of those words.
- A window of k words to the left of Entity 1 and their part-of-speech tags.
- A window of k words to the right of Entity 2 and their part-of-speech tags

3.3.3 Syntactic features

In addition to lexical features authors extracted a number of features based on syntax. In order to generate those features they parsed each sentence with the broad-coverage dependency parser MINIPAR[12]. Syntactic features were mainly consisted of dependency path between two entities and also a 'window node' that was not part of the dependency path. A window node was a node connected to one of the two entities and not part of the dependency path.

3.3.4 Named entity tag features

Most important task of Relation Extraction was finding the entities from the text sentence. So, it would be easy to find relation between those two entities. The authors used the Stanford Name Entity Recognizer of four classes (**Person**,**Location** ,**Organization** and **Others**)

Authors used a multi-class logistic classifier optimized using L-BFGS with Gaussian regularization. Their classifier took an entity pair and a feature vector as input, and returned a relation name and a confidence score based on the probability of the entity pair belonging to that relation. Once all of the entity pairs discovered during testing were classified, all of them were ranked by confidence score and used to generate a list of the n most likely new relation instances.

3.4 Neural Network Model Based Approach

As the no. labelled data can be generated easily using the above approach(i.e. Distant supervision) now most of the work on this field is done by the neural network model. We will go through some of the approaches which are being researched regorously nowadays on Natural Language Processing and as well as Relation Extraction.

3.5 Convolutional Neural Network Based Approach

This paper [5] proposed a novel approach for Relation Extraction using Distant Supervision and sentence level attention and also Entity description. The authors proposed the idea of Multi Instance Learning approach(proposed by [15]) where all sentences containing same entities were placed inside one bag where label of any bag would be those entities connected by the relation in freebase. But the assumption was always not true because entities presented in different sentences might not be always connected by the same relation. To recover from this problem the authors had taken the idea of entity description to get the type of entity. This description improved the model accuracy and also presented a better representation for attention module.

Thus authors proposed the idea of predicting relation class for the unseen bags by extracting features from the bags. The proposed architecture of the aforementioned model is depicted in figure 3.2.

The proposed architecture for the the above model had two sections i.e. Piecewise Convolutional Neural Network (PCNN) Module and also the Sentence level Attention Module.

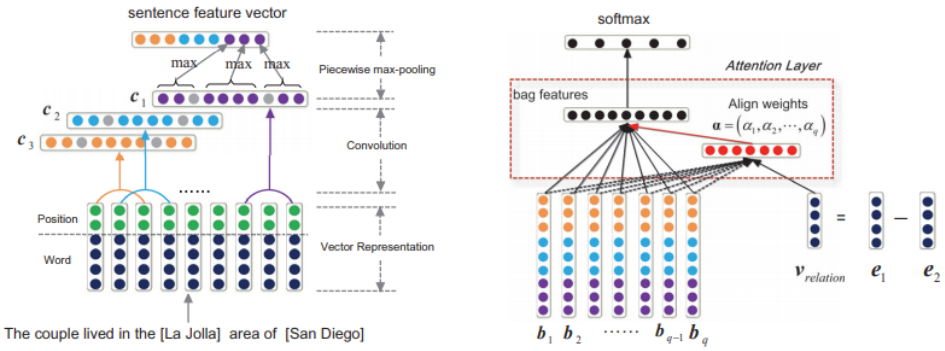


Figure 3.2: PCNNs Module with Sentence Level Attention

3.5.1 PCNNs Module

This module was used to extract feature vector of an instance (sentence) in a bag. For feeding the input into the Neural network model authors used the word embedding methodology([16]) on each sentence to get a lower dimensional vector. Next to detect the entity in the sentence they used the Position Embeddings then output of both the Word Embeddings and Position Embeddings were concatenated to feed into the Neural Network Model.

Then they followed the methodology of Convolutional Neural Network [10]. The input sentence was denoted by S=s_1,s_2,...,$s_{|S|}$ where s_i was the i^{th} token of size $s_i \in \mathbb{R}^k$. Then a filter of size w was used for doing the linear transformation operation. For this the weight matrix of the filter was $\mathbf{W} \in \mathbb{R}^{W \times k}$. Then the output of the above model was generated by computing the equation 3.1.

$$c_j = (\mathbf{W} \cdot \mathbf{S_{j-w+1:j}}) \tag{3.1}$$

Authors used such type of n convolution operation for this they got n no. of convolutional vectors C=c_1,c_2....c_n.

Then authors did the piecewise maxpooling operation on the output of the Convolution Neural Network. Result vector was divided into three parts c_i =($c_{i,1}$,$c_{i,2}$,$c_{i,3}$). From the three vectors authors pool the maximum one by

computing the equation 3.2.

$$p_{i,j} = \mathbf{max}(\mathbf{c_{i,j}}) \tag{3.2}$$

3.5.2 Sentence-level Attention Module:

Then author followed the methodology of the paper [16] by computing the eqution 3.3.

$$v_{relation} = (e_1 - e_2) \tag{3.3}$$

In the above figure(3.2) the vector (b_1,b_2,b_3....bn) were the feature vectors computed by the PCNN module to feed into the attention. They followed the equation proposed by the paper [13].

$$\alpha_i = \frac{\exp(w_i)}{\sum_{j=1}^{q} \exp(w_i)} \tag{3.4}$$

$$w_i = \mathbf{W_a^T}(\tanh[b_i; v_{relation}]) + b_a \tag{3.5}$$

Where b_a is the bias vector. W_a^T is an intermediate matrix α=[α_1,α_2,....,α_n] are the attention weights. Then the bag features were computed using the following equation:

$$b' = \sum_{i=1}^{q} \alpha_i \dot{b_i} \tag{3.6}$$

Then the b' vector was fed into the softmax classifier to get the feature with highest probablity.

Thus authors got the relation vector corresponding to the two entities.

3.6 LSTM Based Approach for Relation Extraction

As we know that LSTM is very much useful for the text data where CNN is very much useful for the image data. The paper [18] presented a novel architecture for End to End Relation Extraction using LSTM on Sequences and Tree Structures.

Authors presented a novel end-to-end model to extract relations between entities on both word sequence and dependency tree structures. Their model allowed jointly modeling of entities and relations in a single model by using both bidirectional sequential (left-to-right and right-to-left) and bidirectional tree-structured (bottom-up and top-down) LSTM RNNs. This model first detected entities and then extracted relations between the detected entities using a single incrementally-decoded NN structure, and the NN parameters were jointly updated using both entity and relation labels. Unlike traditional incremental end-to-end relation extraction models, this model

further incorporated two enhancements into training: entity pretraining, which pretrains the entity model, and scheduled sampling [1], which replaced (unreliable) predicted labels with gold labels in a certain probability. These enhancements alleviated the problem of low-performance entity detection in early stages of training, as well as allowed entity information to relation classification. The architecture proposed by the authors is depicted in figure 3.3.

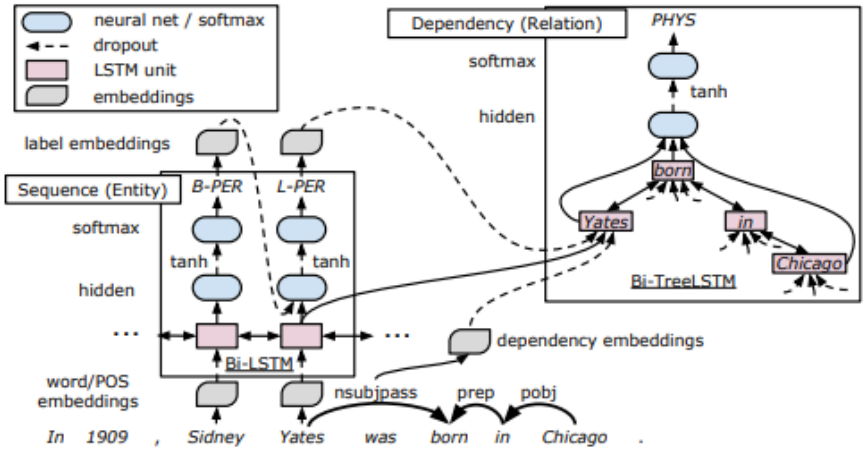


Figure 3.3: Incrementally-decoded end-to-end relation extraction model, with bidirectional sequential and bidirectional tree-structured LSTM-RNNs.

Authors designed model with LSTM-RNNs that represented both word sequences and dependency tree structures, and performed end-to-end extraction of relations between entities on top of these RNNs. The model mainly consisted of three representation layers: a word embeddings layer (embedding layer), a word sequence based LSTM-RNN layer (sequence layer), and finally a dependency subtree based LSTM-RNN layer (dependency layer). During decoding, they built greedy, left-to-right entity detection on the sequence layer and realized relation classification on the dependency layers, where each subtree based LSTM-RNN corresponded to a relation candidate between two detected entities. After decoding the entire model structure, authors updated the parameters simultaneously via back-propagation through time [23]. Then the dependency layers were stacked on the sequence layer, so that embedding and sequence layers were shared by both entity detection and relation classification, and the shared parameters were affected by both entity and relation labels.

3.6.1 Embedding Layer

Authors first made the words,postags,dependency types and also the entity labels into the fixed dimensional vector using one hot representation. Then they fed it into the embedding layer to get the embedded vectors of dimensions respectively v^w, v^p,v^d and v^e.

3.6.2 Sequence Layer

In this layer they followed the methodology of Bi-direction lstm[2]. The LSTM unit at t^{th} took the input vectors as previous hidden state $h_{(t-1)}$, n dimensional input vector $x_{(t-1)}$, input gate input $c_{(t-1)}$ and also the forgate gate followed by the computation of LSTM [2].

$$i_t = \sigma(\mathbf{W^{(i)}} \cdot x_t + \mathbf{U^{(i)}} \cdot h_{(t-1)} + b^i) \tag{3.7}$$

$$f_t = \sigma(\mathbf{W^{(f)}} \cdot x_t + \mathbf{U^{(f)}} \cdot h_{(t-1)} + b^f) \tag{3.8}$$

$$o_t = \sigma(\mathbf{W^{(o)}} \cdot x_t + \mathbf{U^{(o)}} \cdot h_{(t-1)} + b^o) \tag{3.9}$$

$$u_t = \tanh(\mathbf{W^{(u)}} \cdot x_t + \mathbf{U^{(u)}} \cdot h_{(t-1)} + b^u) \tag{3.10}$$

$$c_t = (i_t \odot u_t + f_t \odot c_{(t-1)}) \tag{3.11}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.12}$$

Where $\odot$ is a element-wise multiplication, W and U is the weight matrices and b is the bias vector and σ is an non-linear activation function 3.13.

$$\phi(x) = \frac{1}{1 + \exp^{-x}} \tag{3.13}$$

Similarly tanh is a non-linear activation function 3.14.

$$\tanh(x) = \frac{\exp^{x} + \exp^{-x}}{\exp^{x} + \exp^{-x}} \tag{3.14}$$

.

The LSTM took the input of concatenation of word and it's corresponding pos tags. It gave the hidden matrices as outputs(s_t) which were concatenated to pass it to the subsequent layers.

3.6.3 Entity Detection

Entity detection was done on the top of the sequence layer. For this two layered NN was employed and a softmax output layer was also applied. The equations for the entity detection computation are as follows:

$$h_t^{(e)} = \tanh(\mathbf{W^{(e_h)}} \cdot [s_t; v_{(t-1)}^{(e)}] + b^{e_h}) \tag{3.15}$$

This hidden entity vector was sent to the the softmax classifier to detect the entity with highest probabality.

3.6.4 Dependency Layer

In this section a Bi-directional tree LSTM was applied on the detected entities to find the relation between the detected two entities as the diagram was shown on the above figure3.3 of right side part. To represent the relation between the two entities they followed the two methods:

- Authors primarily employed the shortest path structure **SPTree**, which captured the core dependency path between a target word pair.
- Secondly they also tried with **SubTree** which was nothing but subtree under the lowest common ancestor.
- **FullTree** which was nothing but whole dependency tree of the sentence. It helped to capture context information for the whole sentence.

Then by stacking both the dependency and the sequence layer authors tried to classify relation from the dependency layer's output and detected entities from the entity layer output using the softmax classifier.

Chapter 4

Breif Overview on LSTM and GCN

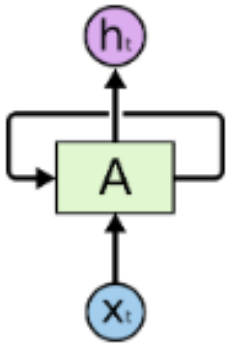
Before going to the architecture that we propose using LSTM and GCN. We will try to give the overview about **LSTM** and **GCN**.

4.1 LSTM

Humans don't start their thinking from scratch every second. As we read any text, we understand each word based on our understanding of previous words. We don't throw everything away and start thinking from scratch again.

Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine we want to classify what kind of event is happening at every point in a movie. It is impossible how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

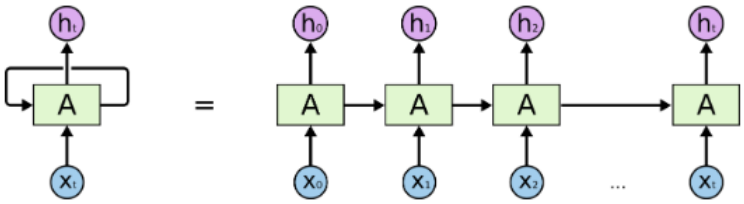
Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, A, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

However, if we think a bit more, it turns out that they are not all different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successors. What happens if we unroll the loop:



This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They are the natural architecture of neural network to use for sequential data.

In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning... The list goes on.

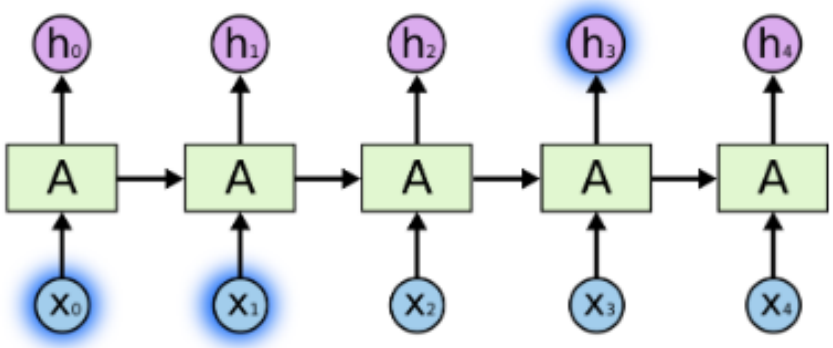
"LSTMs," is a very special kind of recurrent neural network which works, for many tasks, much much better than the standard RNN. Almost all exciting results based on recurrent neural networks are achieved with them.

4.1.1 The Problem of Long-Term Dependencies

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames

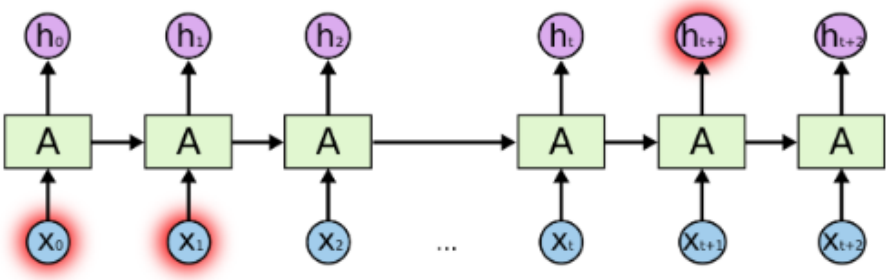
might inform the understanding of the present frame. If RNNs could do this, that will be extremely useful.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the sky," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.



But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France... I speak fluent French." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

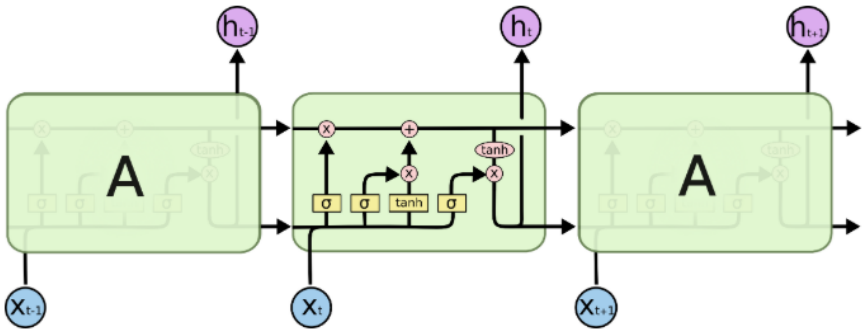
Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



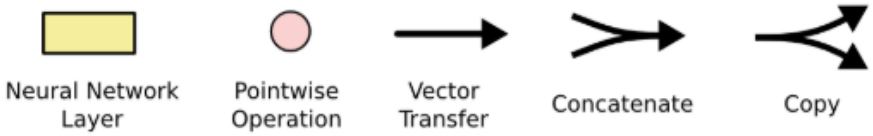
Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by the paper [3].

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior.

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



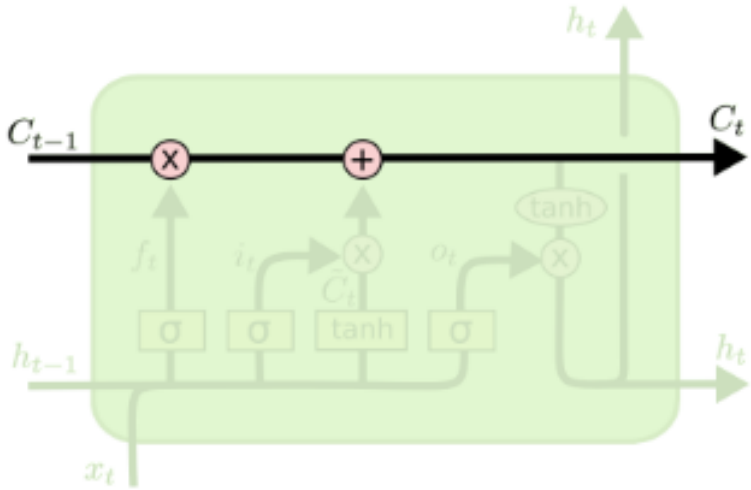
In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.



4.1.2 Idea Behind LSTMs:

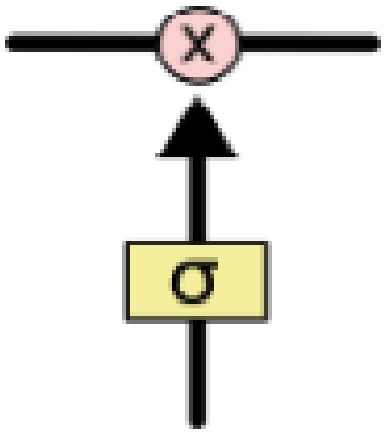
The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

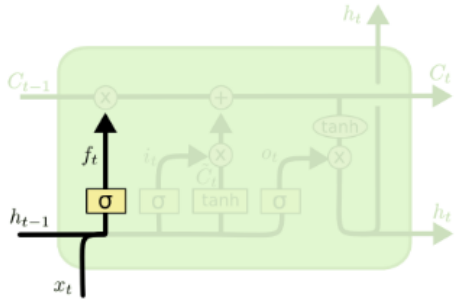
Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through". An LSTM has three of these gates, to protect and control the cell state.

The first step in LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at $h_{(t-1)}$ and x_t, and outputs a number between 0 and 1 for each number in the cell state $C_{(t-1)}$. 1 represents "completely keep this" while a 0 represents "completely forget this."

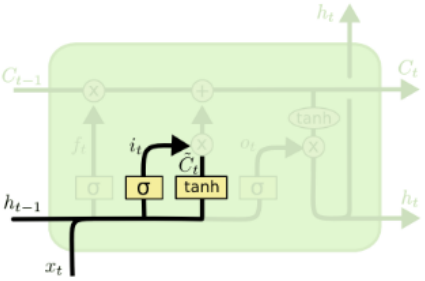
Let's go back to example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The next step is to decide what new information it is going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, C_t, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

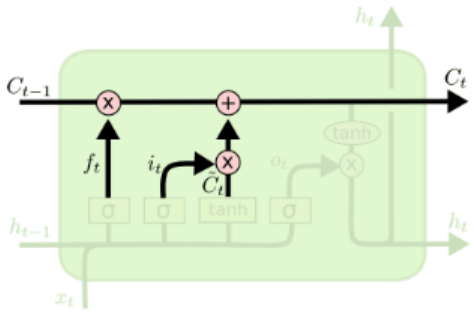


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

It's now time to update the old cell state, $C_{(t-1)}$, into the new cell state $\tilde{C}_t$. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add it C_t. This is the new candidate values, scaled by how much we decided to update each state value.

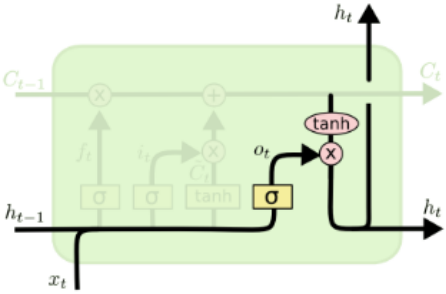
In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into it if that's what follows next.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

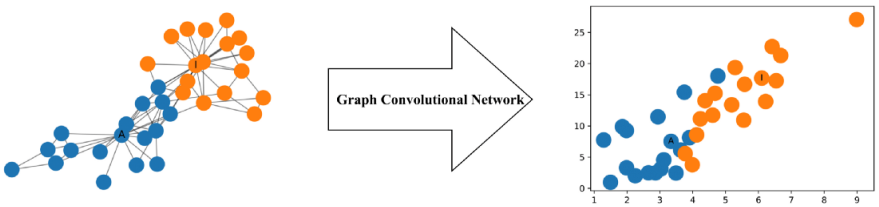
4.2 Graph Convolution Network

As we know that it is difficult to apply Neural Network on graph structure data. This paper[9] gives nice approach for applying the neural network model on the graph structure data.

4.2.1 Definitions

GCNs are a very powerful neural network architecture for machine learning on graphs. In fact, they are so powerful that even a randomly initiated 2-layer GCN can produce useful feature representations of nodes in networks. The figure below illustrates a 2-dimensional representation of each node in a network produced by such a GCN. The relative nearness of nodes in the

network is preserved in the 2-dimensional representation even without any training.



Given a graph G = (V, E), a GCN takes as input ($N \times F^0$) feature matrix (X) where N is the number of nodes and F^0 is the input features of the input matrix (X). As we know that we can represent a graph structure into the adjacency matrix. So, for N nodes graph we will get a adjacency matrix of (N). GCN also takes as input along with the above feature matrix.

A hidden layer in the GCN can thus be written as $H^i = f(H^{(i-1)}, A))$ where $H^0 = X$ and f is a propagation rule. Each layer H^i corresponds to an $(N \times F^i)$ feature matrix where each row is a feature representation of a node. At each layer, these features are aggregated to form the next layer's features using the propagation rule f. In this way, features become increasingly more abstract at each consecutive layer.

So the simplest propagation rule will be as follows:

$$f(H^i, A) = \sigma(\mathbf{A} \cdot \mathbf{H^{(i-1)}} \cdot \mathbf{W^{(i-1)}}) \tag{4.1}$$

Where W^i is the weight matrix at i^{th} GCN layer. σ is a non-linear activation function like ReLU defined as follows:

$$f(x) = \mathbf{max}(\mathbf{0}, \mathbf{x}) \tag{4.2}$$

But the problem with the above approach is if we see the output of the GCN layer. The aggregated representation of the nodes does not include its own features. They only contain the neighbourhood nodes. Only those nodes have self-loop only contain its own features.

Another problem is nodes with large degrees will have large values in their feature representation while nodes with small degrees will have small values. This can cause vanishing or exploding gradients but is also problematic for stochastic gradient descent algorithms which are typically used to train such networks and are sensitive to the scale (or range of values) of each of the input features.

4.2.2 Adding Self Loops

To overcome from the fist problem authors add self-loops with each node by computing the following equation:

$$A' = A + I \tag{4.3}$$

Where A is the adjacency matrix and I is identity matrix.

4.2.3 Feature Normalization:

To overcome from the second approach the author follow the feature normalization by multiplying the inverse degree matrix($D^{-}1$) with adjacency matrix(A'). The computation of the above technique is as follows:

$$f(X, A') = D^{-1} \cdot A' \cdot X \tag{4.4}$$

Thus author gets final representation of all the nodes with added self-loops and the normalized representation of all the nodes.

Chapter 5

Methodology

This chapter presents our methodology for the task. Before going through the methodology part, we first present the Dataset that we use in our system throughout the experiment.

5.1 Dataset Description

We worked with the dataset described in [21]. The dataset is distantly supervised by the Wikidata KnowledgeBase. The dataset was prepared using the Wikipedia data corpus as it is tightly integrated with the Wikidata KB.

First, from each sentence for the complete wikipedia articles the links were extracted; then the corresponding Wikidata entityIDs were retrieved for the linked article. Thus, wikipedia articles were mapped one to one to its corresponding WikidataID. Let us consider the following sentence.

Sachin Tendulkar was born in Mumbai.

The above sentence contains two entities: ***Sachin Tendulkar*** and ***Mumbai***. The corresponding Wikidata entityIDs are extracted as ***Sachin Tendulkar*** $\rightarrow$ **Q9488** and ***Mumbai*** $\rightarrow$ **Q1156**.

Then the noun chunks and entities are extracted using chunking and the Standford CoreNLP [14] toolkit which were not covered by the wikipedia annotation. Then for those chunks and entities, the authors manually extracted the Wikidata entity IDs. For entities like DATE, authors used the HiedelTime [22] annotation. For the given entities in the input sentences, they queried the Wikidata Knowledge Base. If the entities are found in the KB (Wikidata) then the relation that connects the entities in KB was replaced in the input sentence using the concept of Distant Supervision. Then they filtered out those sentences which have less than 3 entities.

Finally 353 relations were extracted. Total 8.5 lakh sentences were dumped into the json file. Each sentence was tokenized and there was an

"edgeSet" which contains the entity pairs and the corresponding relations. EdgeSet contains multiple entity pairs and one relation from the given relationset. The training dataset was prepared using 3.5 lakhs sentence, the validation data was prepared using 1.5 lakhs sentences and the test data was prepared using 3.5 lakhs sentences.

Thus the dataset was prepared using distant-supervision methodology.

5.2 LSTM Attention based Approach

The main aim of the relation extraction task is to extract relation between the two target entities from the given sentence. We apply the methodology of the paper [21] where the process of extracting relations is as follows. In the above paper [18] we have discussed that LSTM can be applied on the relation extraction task from the given sentence. Here using the attention mechanism we try to capture the relation between the target entities using the context information. Where previous approaches neglect the context information. The following example illustrated the idea.

Shonar Kella *is a 1971* ***mystery novel*** *by Bengali writer and filmmaker* ***Satyajit Ray***.

In the above sentence, the words in bold fonts are the entities extracted during the data preparation step. To identify the relation (**DIRECTED_BY**) between the entities **Shonar Kella** and **Satyajit Ray**, it is important to classify the relation (**INSTANCE_OF**) between the entities **Shonar kella** and the **mystery novel**.

To argue the above proposal, we have proposed a Bi-LSTM mechanism with attention which outperforms the the methodology of extracting relation using only LSTM encoder and the methodology of LSTM attention.

5.3 Architecture

We know that in text the length of the sentences varies widely. To overcome this problem we take fixed size of sentence length (n) to get fixed size vector for each sentence.

We map each token in a sentence to d dimensional word embedding vector using a word embedding matrix $\mathbf{W} \in \mathbb{R}^{|V| \times d}$, where V is the vocabulary size.

Then we use position embedding to make the model understand about the entities. The process is described below. Each token in the sentence either belongs to entity e_1 or entity e_2 or neither of these two. A marker embedding is randomly initialized with the dimension of $P \in \mathbb{R}^{(3 \times d)}$. Then we concatenate both the word-embeddings (W_n) and position embeddings(P_n). The concatenated vector is $[W_n, P_n]$.

Then, we at first fit the concatenated vector into the LSTM [3] relation encoder to make our baseline model. We get the output($o_s \in \mathbb{R}^o$). Then we classify the output of the LSTM output vector (o_s) by the softmax classifier using the following equation.

$$P(r| < e_1, e_2 >) = \frac{\exp(f_r)}{\sum_{i=1}^{n_r} \exp(f_i)} \tag{5.1}$$

$$f_i = \mathbf{y_i} \cdot \mathbf{o_s} + \mathbf{b_i} \tag{5.2}$$

Where y_i is the weight vector and the b_i is the bias vector. Thus we get a classified relation between the target entities.

Next, we follow the methodology of attention mechanism proposed by [21]. To detect the relation between the target entities, we take other entity-pair relations which is known as context relation of the sentence. Similarly, we make concatenation of the word and position marker. Then we jointly feed the target entities corresponding to its position marker and the context entities corresponding to its the position markers individually into the LSTM encoder. We get the individual output vector (o_s) for target entities and output vectors corresponding to its context entities (o_c). All the required hidden matrices for both the target relation and context relations are jointly learned during the training of LSTM encoder. So, we get output vector (o_i) for each individual context relation.

We sum up all the relation outputs (o_i) using the following equation5.3 where m is the number of context relations.

$$o_c = \sum_{i=0}^{m} o_i \tag{5.3}$$

5.3.1 Attention Mechanism

Then we apply the attention mechanism i.e. the weighted context-sum during summing up all the context relations (**??**). We use the following equation for doing the above computation to apply the attention mechanism where a_i is the attention weight at the i^{th} layer.

$$o_c = \sum_{i=0}^{m} a_i \cdot o_i \tag{5.4}$$

The attention mechanism follows the probability distribution i.e. which should be useful or not. The attention mechanism computes the following equation to update the weights properly where $f(o_i, o_s) = o_i w o_s$, w is the attention weight matrix that is learned during training operation.

$$a_i = \frac{\exp(f(o_i, o_s))}{\sum_{j=0}^{m} \exp(f(o_j, o_s))} \tag{5.5}$$

5.3.2 Classifying Output Relation

We get a final output of the context relation (o_c) using the attention mechanism. Then we concatenate both the targeted output (o_s) and the context output (o_c). Finally we feed the $o = [o_s, o_c]$ into the softmax classifier (5.1) to predict the final relation between the two target entities. Our modified proposed architecture of the same proposal using the Bi-LSTM encoder is depicted in Figure 5.2.

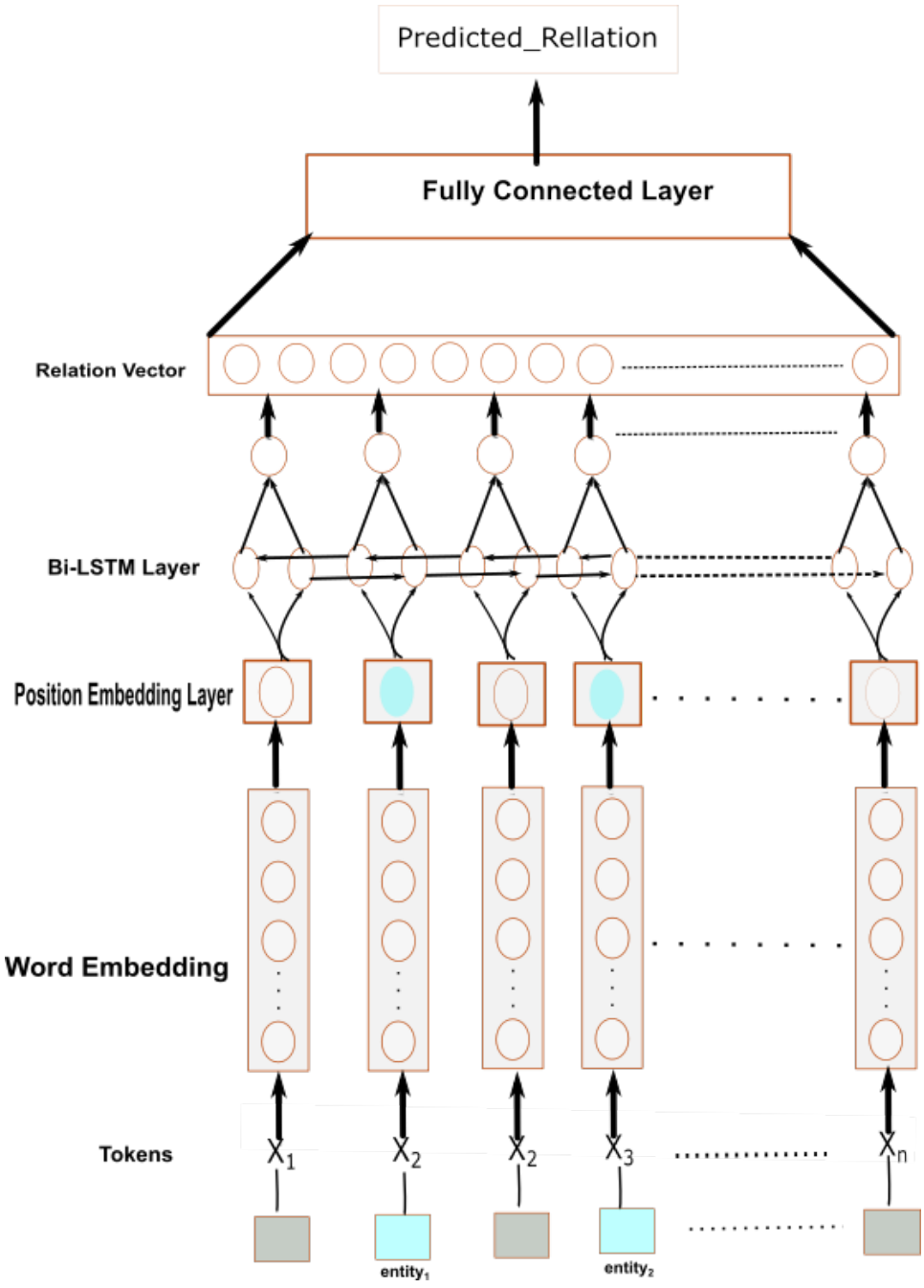


Figure 5.1: Relation Encoder using Bi-LSTM

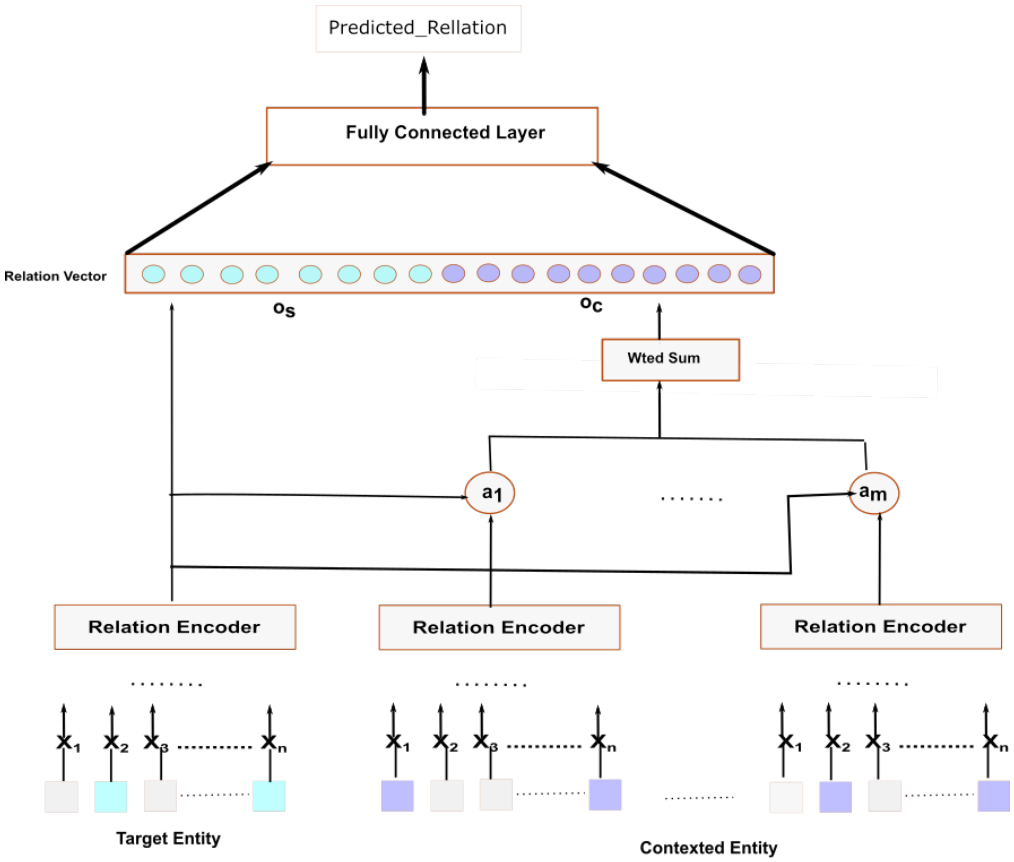


Figure 5.2: Relation Encoder using Bi-LSTM attention Mechanism

We also try the same mechanism of attention based encoder as represented in the Figure 5.2 for classifying the relation between the two targated entities in the given sentence using the **Bi-LSTM**. We follow the methodology that is described above; instead of LSTM encoder, we just use Bi-LSTM encoder. The results of Bi-LSTM attention based approach outperforms the results of both LSTM and LSTM with attention. Bi-LSTM results beat all the previous results that can be found in the literature for Relation Extraction.

5.4 Graph Convolution Based Approach

Besides experimenting with the above mentioned three approaches, we also propose a novel architecture using **Graph Convolution Network**.

5.4.1 GCN based Architecture for Relation Extraction

We first implement the relation extraction task using GCN following the methodology of [9]. Our proposed architecture diagram is presented in Figure 5.3.

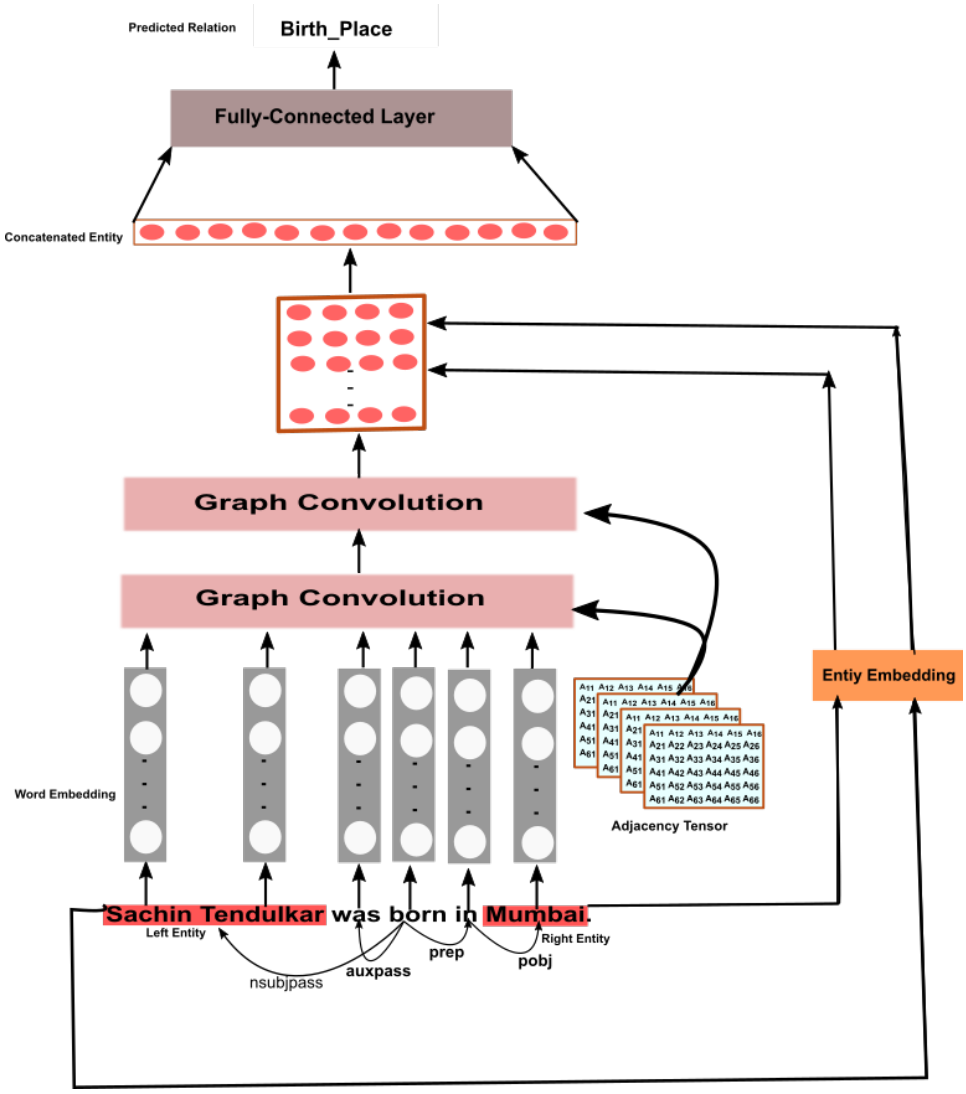


Figure 5.3: Relation Extraction using GCN Mechanism

For each sentence, we prepare a typed adjacency tensor (A) using dependency graph (i.e. considering dependency graph as an undirected graph). Let R be the set of all the dependency relations and n be the maximum length of any sentence. So, for each relation in set R, we can represent any sentence with $(n \times n)$ dimensional adjacency matrix where A_{ij}=1 if there is an edge between the i^{th} node and the j^{th} node. We add a self loop to each

of the node so that it can contain its own features. We get an adjacency tensor (A) of shape $A \in \mathbb{R}^{R\times n\times n}$.

We map each token in a sentence to d dimensional word embedding vector using a word embedding matrix $\mathbf{W}\in \mathbb{R}^{|V|\times d}$,where V is the vocabulary size.

Then we apply a feature normalization of the adjacency matrix using the inverse degree matrix ($D^{-1}\in \mathbb{R}^{n\times n}$) of each sentence by computing the following equation 5.6.

$$A' = (D^{-1} \cdot A) \tag{5.6}$$

Then we follow the methodology of Graph Convolution Network [9] as we discussed earlier by computing the following propagation rule followed by max pooling operation 5.7.

$$O^{(i)} = maxpool(f(A'^{(i)} \cdot W^{(i)} \cdot H^{(i)})) \tag{5.7}$$

Here H^i is the hidden matrix at i^{th} layer, $W^{(i)}$ is the linear transformation at i^{th} layer, the $A'^{(i)}$ is the normalized adjacency input tensor at i^{th} layer and f is a nonlinear activation function (4.4). O^i is the output tensor at the i^{th} layer.

We also represent a set of entity tokens (created during the time data generation steps) for each sentence to a fixed length (n) one-hot representation. We get two entity vectors for each sentence i.e. left entity vector (e_l $\in \mathbb{I}^{1\times n}$) and also right entity vector ($e_r \in \mathbb{I}^{1\times n}$).

Using the output tensor ($O^{(i)} \in \mathbb{R}^{n\times d}$), we compute the following equation 5.8 for both the right and left entity vector. We get left entity vector ($e'_l \in \mathbb{R}^{d\times 1}$) and right entity vector ($e'_r \in \mathbb{R}^{d\times 1}$).

$$e' = e \cdot O \tag{5.8}$$

Thus we convert the simple entity representation to an aggregated feature representation. We concatenate both the left and right entity representation. We pass the final entity node representation (e'') through a fully connected layer with non-linear activation function softmax to map it to its corresponding relation vector with the highest probability. Thus we get our final output relation vector (O_r) by computing the following equation where y_i is the weight matrix and b_i is the bias vector.

$$P(\mathbf{O_r}|\mathbf{e''}) = \frac{\exp(f_r)}{\sum_{i=1}^{n_r}\exp(f_i)} \tag{5.9}$$

$$f_i = \mathbf{y_i} \cdot \mathbf{e''} + \mathbf{b_i} \tag{5.10}$$

Thus by applying the aforementioned model, we set up a baseline architecture. Next we discuss another method of GCN to improve over the baseline model.

5.4.2 Graph Convolution Network with Latent Adjacency

The above approach remains same with just one extra latent adjacency tensor with original adjacency input tensor. To build the adjacency tensor of any sentence we take the dependency graph using the spacy [4] toolkit. We know that no toolkit is always perfect. For this reason we randomly initialize another latent adjacency tensor ($A_L \in \mathbb{R}^{R\times n\times n}$) with the same shape of adjacency tensor (A). So, both the hidden matrix and the latent adjacency tensor will be learned during the training of GCN. Our model architecture is depicted in figure 5.4.

We follow the above computation with slight modification. First, we add both the adjacency tensor and the latent adjacency tensor. So, here we are following one type of smoothing technique. Nodes that do not get any feature value will get some random weight which will be tuned during the training. We do the above computation by computing the following equation 5.11.

$$A_u = A + A_L \tag{5.11}$$

From here we get a latent adjacency matrix with shape ($A_u \in \mathbb{R}^{R\times n\times n}$).

Then we follow the above methodology in the following way. For normalizing the feature we follow the equation (5.6) with the new adjacency matrix.

$$A'_u = D^{-1} \cdot A_u \tag{5.12}$$

Similarly we follow the methodology of GCN [9] following max-pooling operation by computing the equation below where $O_u^{(i)}$ is the output vector at i^{th} layer, ${A'_u}^{(i)}$ is the normalized tensor at i^{th} layer,f is the non-linear activation function ReLU activation.

$$O_u^{(i)} = maxpool(f({A'_u}^{(i)} \cdot W_u^{(i)} \cdot H_u^{(i)})) \tag{5.13}$$

Then, we also represent a set of entity tokens (created during the time of data generation steps) for each sentence to a fixed length (n) one-hot representation. We get two entity vectors for each sentence i.e. left entity vector ($e_l \in \mathbb{I}^{1\times n}$) and also right entity vector ($e_r \in \mathbb{I}^{1\times n}$).

Using the output tensor ($O_u^{(i)} \in \mathbb{R}^{n\times d}$) we compute the following equation 5.14 for both the right and left entity vector. We get left entity vector ($e'_l \in \mathbb{R}^{d\times 1}$) and right entity vector ($e'_r \in \mathbb{R}^{d\times 1}$).

$$e'_u = e_u \cdot O_u \tag{5.14}$$

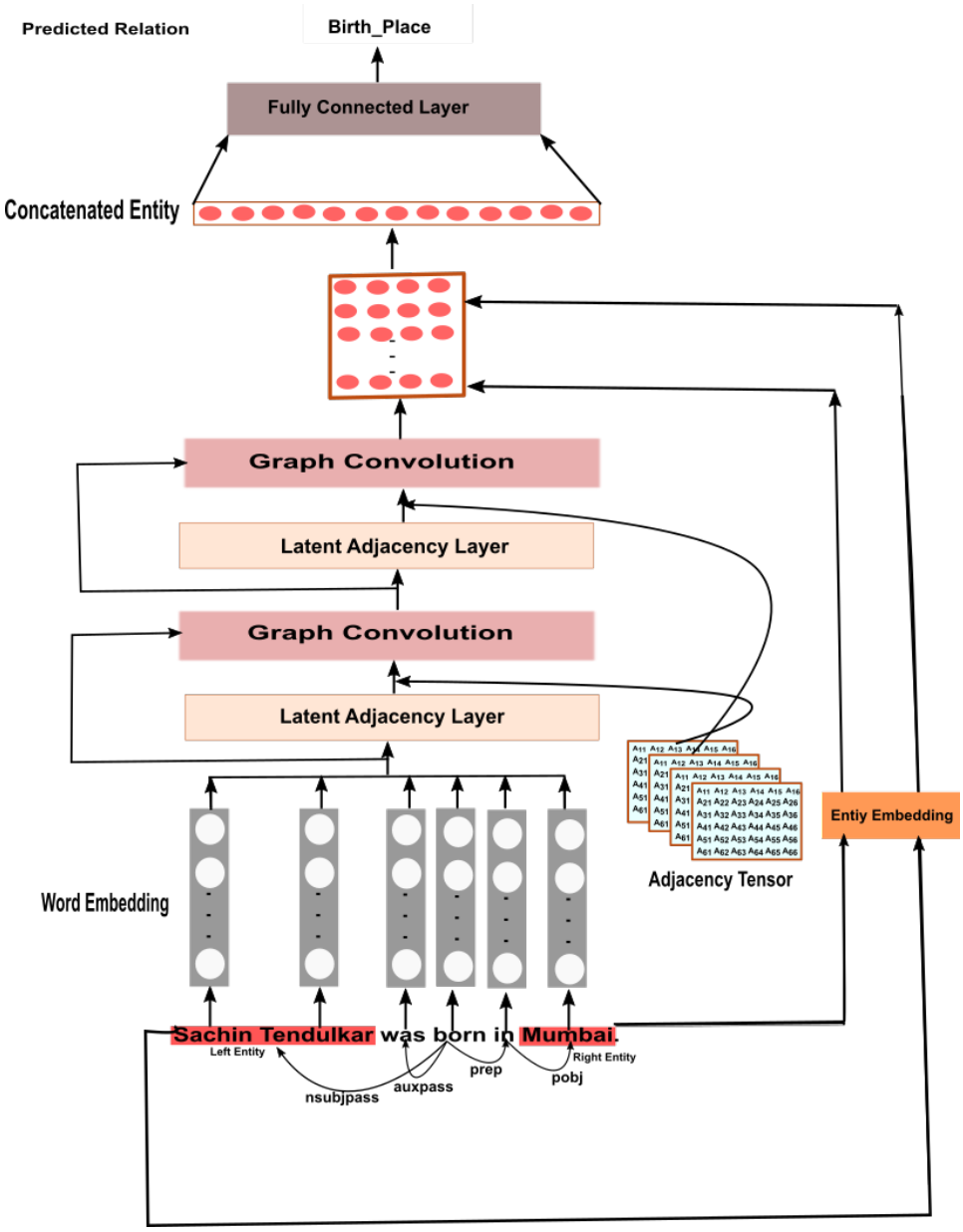


Figure 5.4: Relation Extraction using GCN with Latent Adjacency Representation

where e'_u is the output vector of entity representation. Thus, we convert the one-hot representation into the normalized feature representation. We concatenate both left and right entity matrix.

Then we push those entity matrix representation into the fully-connected layer with non-linear activation function (5.9) to map each entity matrix to its corresponding relation output vector.

Chapter 6

System Description

We will discuss about our system description about the above models.

6.1 LSTM based System Description

In the above section we said that we mapped each token to a fixed dimensional word-embedding vector. To embed the word token we use 50 dimensional Glove [19].

Throughout the experiment we use position embedding markers of dimension 3.

Finally we concatenate both the output of the word-embeddings and also the output of the postition embedding to push them into the LSTM or Bi-LSTM.

After going through the dataset(i.e. training data and validation data) we observe most of the sentence length lies below 37. For this we set the length(l) 36. We ignore other sentences of length more than 36. We apply post padding operation with zeros for those sentences which have length less than 36.

Throughout the experiment we use dropout. We apply dropout of .5 on the embedding layer. We also apply it on the fully-connected layer(i.e. Dense layer) for extracting relation from the given two entities in the given text.

We apply regularization in the position marker embedding layer to solve the problem of overfitting by penalizing the weights as per requirement during training. We use (l_2) regularization throughout the experiment.

As we care about the context relation to predict the target relation between the target entities as mentioed above in the methodology chapter. We at a time take 7 entity-relation pairs per sentence. According to the above condition we split our dataset i.e. if there are more than 7 entity-relation pairs per sentence we split it accordingly.

We use Keras to implement the system using python.

Throughout the experiment we use 'Categorical Crossentropy' as a loss function and Adam [8] as an optimizer function.

6.1.1 Baseline Implementation for LSTM

First we represent each sentence into the one-hot representation using the index number of glove word-embeddings. Then we feed the that representation into the input layer. As we need to repeat each sentence 7 times we apply repeatvector(7) on the input layer output.

Then we take the above output of the input layer into the embedding layer. The output from the embedding will be 50 dimension.

Then we represent the entity input into one-hot representation using the following methodology. We mark entity tokens '1' if it present left side of the relation in the given triplet. Similarly, for other entity tokens which present in the right side of the triplets we mark those tokens with '3'. Otherwise all the tokens other than the above two genres we mark them as '3'.

Thus we represent the entity input representation using the above methodology. We feed them into another embedding layer with output dimension 3.

We concatenate both the word-embeddings layer output and the entity-marker embedding layer output to get a '53' dimensional vector.

We apply both the dropout and regularization using the aforementioned technique.

Next we feed the concatenated input representation of dimension 53 into the LSTM cell. From the above cell we get hidden entity-token representation. Finally, we try to map them to their corresponding relation vector (which is represented by binary one-hot representation of size 353 dimension). Mapping was done using the fully-connected layer of dimension 353 with an non-linear activation function of softmax.

6.1.2 LSTM and Bi-LSTM Attentionbased

We apply the same methodology of the above one with slightly some modification. In the case of the Bi-LSTM we put the concatenated representation (53 dimension) into Bi-LSTM cell of keras. Bi-LSTM works similary as LSTM disscussed in the chapter 4 with one modification. Bi-LSTM not only remember the previous sequence information to predict the next sequence information it also cares about the future sequence information.

After putting the concatenated representation into LSTM and also on Bi-LSTM we apply the attention mechanism for both the above architectures respectively. Finally we successfully connect the hidden representation into a fully-connected layer using Adam optimizer and softmax as an activation function. Thus we get the output of relation vector among 353 relation instances between two target entities using the contextual information. We

train the above systems using training data (as discussed previously) and validation data. We test the system using the test dataset.

We train our above three system in google colaboratory environment with 50 epochs and batch size of 256.

6.2 GCN based approach

As previously discussed in chapter 4. We build the typed adjacency tensor using the adjacency relation set of spacy dependency relations. In spacy toolkit the number of dependency relations for English language is 48. We set the sentence length to 36. So, we make one adjacency matrix for each relation from the relation set R. Thus we get an adjacency tensor of size $(48 \times 36 \times 36)$ for each sentence. We make each adjacency matrix added with self-loop.

We build a degree layer to get the degree representation of all the nodes in each sentence.

We put the output of the degree layer to the Multiplication layer to compute the above computation (5.6). Thus we get a normalized adjacency tensor representation where all the nodes of that tensor is normalized by multiplying with its corresponding inverse degree matrix.

First we make one-hot representation of each sentence using the word index of Glove word-embedding of 50 dimension. We feed this one-hot representation into a Embedding layer to get the the embedded output vector for each sentence.

Next we build a graph layer which will take input of both Embedding layer output and the normalized representation of adjacency tensor. In the graph layer we randomly initialize a weight matrix which will be trained during the training time of this whole model.

Graph layer internally do the computation of (5.7) to get the Dense representation of each node (i.e. this layer will return each node representation which contains it's corresponding neighbor nodes and also itself).

We apply another Graph layer which will take input adjacency tensor (A') and the output of the first Graph Layer. So this layer will return another more dense hidden representation of each node.

We take each entity relation pair and we represent entity tokens into one-hot representation. We take left entity words and we make map them into a vector representation of length 36. If the entity tokens present in its corresponding sentence then we make value 1 for that particular position in the vector. otherwise we make 0 for rest of all the positions. We apply similar approach for the right entity tokens. We create similarly right entity vector using the same concept.

Then we build entity layer to do the computation mentioned in (5.8). So from the entity layer we get a left entity matrix and right entity matrix rep-

resentation with it's corresponding nodes for each sentence. We concatenate both the entity representation.

Now we feed both the left entity and right entity to fully-connected layer of 353 dimension with softmax (5.1) activation function. Thus we predict the target relation corresponding to its target entities.

We train this model with batch size 64 with 30 epochs.

6.3 Latent GCN based approach

As the architecture is almost same as the above GCN with one modification. We have already discussed about it. Now we will give system description of our proposed approach.

For latent adjacency tensor we make a Noise layer. It will give us a latent adjacency tensor representation with randomly initialized weights. In this layer we push adjacency tensor(A) and it returns output latent adjacency tensor representation by computing the equation 5.11. The output shape(A_u) will be of dimension ($48 \times 36 \times 36$). Here 48 is the number of dependency relation of Spacy for English and 36 is the maximum length of sentence as discussed in the above section. This layer computes the computation of equation (**??**) using sigmoid as an activation.

Then we push this updated adjacency tensor representation to a Degree Layer to get corresponding inverse degree matrix representation (D^{-1}) of each sentence. Then we feed both the latent adjacency tensor(A_u) and the Degree matrix representation(D^{-1}) into the Multiplication layer to normalize the feature representation.

So we get a normalized latent adjacency tensor representation (A'_u). Then we take both the normalized adjacency tensor(A'_u) and word embedding tensor representation as inputs into one Graph Layer.The computation for this layer follows the equation (5.13) using relu (4.4) as an activation function. We get A_1 as output which is a dense representation of all the nodes (i.e. here each node covers its all the neighbor nodes with degree 1).

Then we repeat the same procedure one times in the same way. We push the output of the graph layer to the same noise layer to smooth all the nodes properly. This layer will give us latent adjacency tensor representation using the (**??**) and sigmoid activation function.

Similarly we get its corresponding inverse degree matrix (D^{-1}) and normalized latent adjacency feature representation(A'_u) using both the Degree Layer and Multiplication Layer. The Multiplication Layer compute equation (5.6).

Then we take A_1 and A'_u to feed into the another graph layer with same computation (5.13) and relu(4.4) as an activation. Similarly we get A_2 more dense representation of all the nodes.

Now we take one-hot representation of left and right entity vector. We

take left-entity vector to the Entity Layer and right-entity vector to the same Entity Layer. So, we represent both the left and right entity vector to its corresponding node representation using the computation of equation (5.14).

We concatenate both left and right entity representation. We feed it into the fully-connected layer to map with its corresponding relation vector using the softmax(5.1).

We train this model with batch size 64 with 30 epochs.

Chapter 7

Results

The experimental results that we have obtained from the aforementioned models are presented in this chapter. Earlier we mentioned that among the aforementioned 5 approaches, Bi-LSTM performs very well. This model provides an accuracy of about 82%. Our proposed approach (GCN with Latent Adjacency Tensor) results in accuracy near about 70% on the same dataset. We use four metrics to evaluate our models - accuracy, precision, recall and f1-score. The metrics as defined as given below.

$$Accuracy = \frac{\text{Total no. of correct answers returned by the System}}{\text{Total No. of Answers}} \tag{7.1}$$

$$Precision = \frac{\text{No. of correct answers returned by the system}}{\text{No. of answers returned by the system}} \tag{7.2}$$

$$Recall = \frac{\text{No. of correct answers returned by the system}}{\text{No. of answers in gold standard answer}} \tag{7.3}$$

$$F1 - score = \frac{2 \times Precision \times Recall}{\text{Precision + Recall}} \tag{7.4}$$

7.1 LSTM approach based Result

The results of the LSTM based models are presented in Table 7.1. From the results obtained acrorss the four evaluation metrics, it can be observed that the Bi-LSTM model performs remarkably well.

Model	Acc.	Precision	Recall	F1-Score
Bi-LSTM + Att.	0.819	0.817	0.796	0.807
LSTM + Att.	0.805	0.799	0.776	0.787
LSTM	0.721	0.792	0.695	0.740

Table 7.1: Results of LSTM based models

7.2 GCN approach Based Result

This section presents the results of our proposed GCN based approach. The results of 2 separate experiments, with and without Latent Adjacency tensor, are reported in Table 7.2. It can be observed from the table that the proposed GCN model provides 64.3% accuracy and .630 F1-score, while adding the latent adjacency tensor to the GCN model, provides some improvements and it results in 68.8% accuracy and .690 F1-score.

Model	Acc.	Precision	Recall	F1-Score
GCN	0.643	0.650	0.640	0.630
GCN + Latent Adj.	0.688	0.700	0.690	0.690

Table 7.2: Results of GCN based models

Chapter 8

Conclusion and Future Work

This thesis presents a work on extracting knowledge from unstructured text. Initially we tried with some rule-based approach for the knowledge extraction task. However, it was too much tedious to write down all the rules for the task of Relation Extraction. For this reason we shifted to the neural network based approaches. We explored a total of 5 approaches - LSTM, LSTM with attention, Bi-LSTM with attention, our novel GCN and GCN with latent adjacency tensor.

Bi-LSTM with attention mechanism performs well on the aforementioned data-set. Among the 5 approaches, accuracy of our novel architecture (GCN with Latent Adjacency Tensor representation) does not outperform the Bi-LSTM model. So, we will try to modify this architecture as future work so that we can combine the strengths of Bi-LSTM and GCN in a combined framework. These above approaches can also be applied on any other type of relation extraction task.

In future, instead of generating adjacency matrix from the dependency graph, we will try to learn it automatically through neural network. We will feed each sentece vector and its corresponding word-embedding matrix into the LSTM. The output that the LSTM will generate will be nothing but a hidden representation of the adjacency matrix and we will put that output into the GCN to classify the relation class. As we have not applied our approaches on other available data-sets, we will try our approaches on other datasets so that we can conclude more concretely. In future we will also try to increase the number of relation classes which will help to infer new relations between entities in a given text more accurately. This will help to populate Knowledge Bases.

Bibliography

[1] Samy Bengio et al. "Scheduled sampling for sequence prediction with recurrent neural networks". In: *Advances in Neural Information Processing Systems*. 2015, pp. 1171–1179.

[2] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[3] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[4] Matthew Honnibal and Ines Montani. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing". In: *To appear* (2017).

[5] Guoliang Ji et al. "Distant supervision for relation extraction with sentence-level attention and entity descriptions". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[6] Nanda Kambhatla. "Combining Lexical, Syntactic, and Semantic Features with Maximum Entropy Models for Extracting Relations". In: *Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions*. ACLdemo '04. Barcelona, Spain: Association for Computational Linguistics, 2004. DOI: 10.3115/1219044.1219066. URL: http://dx.doi.org/10.3115/1219044.1219066.

[7] Natthawut Kertkeidkachorn and Ryutaro Ichise. "An automatic knowledge graph creation framework from natural language text". In: *IEICE TRANSACTIONS on Information and Systems* 101.1 (2018), pp. 90–98.

[8] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[9] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[11] Jens Lehmann et al. "DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia". In: *Semantic Web* 6.2 (2015), pp. 167–195.

[12] Dekang Lin. "Dependency-based evaluation of MINIPAR". In: *Treebanks*. Springer, 2003, pp. 317–329.

[13] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421. DOI: 10.18653/v1/D15-1166. URL: https://www.aclweb.org/anthology/D15-1166.

[14] Christopher Manning et al. "The Stanford CoreNLP natural language processing toolkit". In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, pp. 55–60.

[15] Oded Maron and Tomás Lozano-Pérez. "A framework for multiple-instance learning". In: *Advances in neural information processing systems*. 1998, pp. 570–576.

[16] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[17] Mike Mintz et al. "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: Volume 2-Volume 2*. Association for Computational Linguistics. 2009, pp. 1003–1011.

[18] Makoto Miwa and Mohit Bansal. "End-to-end relation extraction using lstms on sequences and tree structures". In: *arXiv preprint arXiv:1601.00770* (2016).

[19] Jeffrey Pennington, Richard Socher, and Christopher 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.

[20] Rion Snow, Daniel Jurafsky, and Andrew Y Ng. "Learning syntactic patterns for automatic hypernym discovery". In: *Advances in neural information processing systems*. 2005, pp. 1297–1304.

[21] Daniil Sorokin and Iryna Gurevych. "Context-Aware Representations for Knowledge Base Relation Extraction". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Copenhagen, Denmark: Association for Computational Linguistics, 2017, pp. 1784–1789. DOI: 10.18653/v1/D17-1188.

[22] Jannik Strötgen and Michael Gertz. "Multilingual and cross-domain temporal tagging". In: *Language Resources and Evaluation* 47.2 (2013), pp. 269–298.

[23] Paul J Werbos et al. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.