Knowledge Graph: Overview, Embeddings and its Application in Question Answering

Harsh Peswani
203050043

Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai, 400076
2021

Under the guidance of Prof. Pushpak Bhattacharyya.

Acknowledgement

I would like to thank my guide **Prof. Pushpak Bhattacharyya** for always pointing me towards the right direction, and always helping me whenever I got stuck. I would also like to thank my seniors Prince Kumar and Sovan Kumar Sahoo, who always suggested me good papers to read. I would also like to thank all the members of CFILT LAB, IIT Bombay for always helping me.

Abstract

We live in the information age; we have lots of information. It is hard for the algorithms to use information in the unstructured format. Structured data are easy to interpret, one way of structuring data/information is through Knowledge Graphs. There are many applications of Knowledge Graphs for example, Question Answering. Search engines like Google also uses Knowledge Graph.

Knowledge Graph is structured representation of the data, still it is difficult for the machines to use them directly. Machine learning algorithm cannot understand words, it can only understand numbers. Knowledge Graph embeddings can be used for this purpose, knowledge graph embedding algorithms tries to capture all the information in knowledge graph into the vectors. Machine learning algorithms can use these vectors for increasing their performance.

This seminar report describes various existing Knowledge Graphs. It also describes Knowledge Graph Embeddings in detail. One of the important applications of Knowledge Graph Embedding is Question Answering, which is also discussed. Some algorithms of Question Answering are also described in this report. A brief overview of the Joint IIT LG Project (Question Answering Using Knowledge Graph) is also discussed in this report.

Contents

List of Figures

Chapter 1

Introduction and Motivation

This chapter gives an overview of Knowledge Graph, Knowledge Graph Embeddings, and Question Answering. The information which is needed before diving deep into these topics is given in this chapter. Section 1.1 gives a small introduction on Knowledge Graph, explains Knowledge Graph using an example. Section 1.2 explains knowledge graph embeddings, gives an intuition of knowledge graph embeddings through an example. Chapter 4 contains the Application of Knowledge Graph Embeddings in Question Answering therefore section 1.3 gives a very small introduction on Question Answering which should be sufficient to understand chapter 4. Finally, section 1.4 contains the organization of the report.

1.1 Knowledge Graph

We live in the information age, we have a lot of unstructured information. While the unstructured data have many applications, it is difficult to do reasoning from them. Structured data are easy to interpret and it is easy to do reasoning from them.

Knowledge Graph is one of the many ways to represent data in a structured manner. Knowledge graph is a graph so it will contain nodes and edges. Nodes represent entities and edges represent the relation between them. Knowledge graph usually consists of classes, individuals, object properties, data properties, and rules (Limdiwala et al. 2019). Figure 1.1 gives an example of a knowledge graph (Limdiwala et al. 2019).

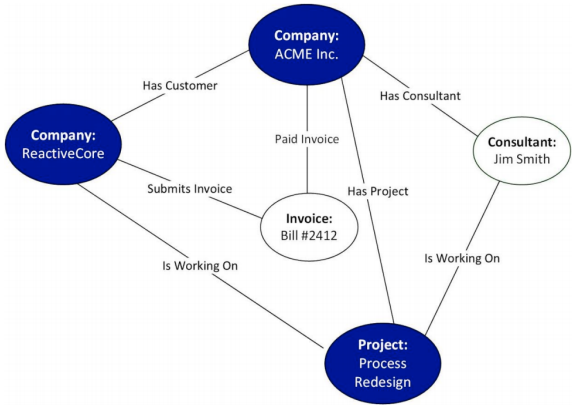


Figure 1.1: Example of Knowledge Graph

The class represents kinds of things, for example, Company. Individuals represent a specific member of the class, for example, ReactiveCore. Object property is the relationship between two individuals for example Has Customer is a relationship between ReactiveCore and ACME Inc. Data property is a relationship between an individual and a constant property for example Submits Invoice in the above example. The rules are facts for example sun rises in the east.

Google first used the term Knowledge Graph in 2012 for the semantic web created by them. Google search engine uses this Knowledge Graph. The information which is retrieved from the knowledge graph is shown in infobox. Figure 1.2 shows an example of the info box.

Figure 1.2: Example of Info Box

1.2 Knowledge Graph Embedding

Embedding is a vector representation of things(word, image, video) in low-dimensional space. Algorithm cannot understand or interpret things(word, image, video) as humans do for example we know the meaning of the word dog but the algorithm does not know. We can use embeddings so that algorithm understands/interprets the meaning of things.

Word embeddings are very useful in NLP tasks. Word embeddings are based on the

distributional hypothesis which suggests that if two words are semantically similar then they should be more distributionally similar ("you shall know a word by the company it keeps" - Firth). Similar words have similar representations for example dog and cat are similar, table and lion are not similar. This type of semantic properties are captured by the word embeddings.

Similar to word embeddings there are knowledge graph embeddings that try to capture whole knowledge graphs in the form of vector representation. Intuition is also very similar to word embeddings, similar entities(nodes) should have similar representation. Similar entities are those which have similar neighbors (which are connected by similar relations).

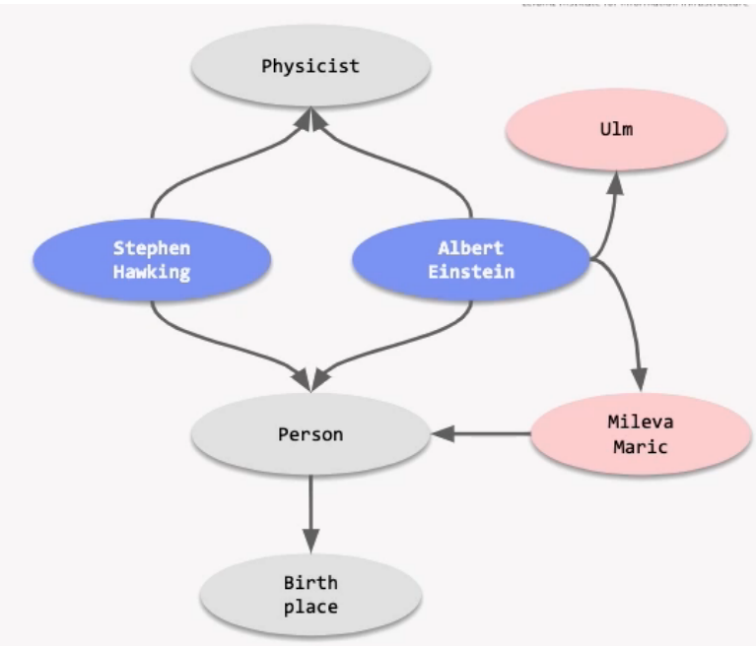


Figure 1.3: Intuition Behind Knowledge Graph Embedding

In Figure 1.3 (Open HPI KG Course) Stephen Hawking and Albert Einstein are similar nodes as they are connected with similar neighbors (Physicist and Person), thus they will have similar vector representation.

We can calculate these knowledge graph embeddings by many methods some of them are:-

- **Translation Based Embeddings:** TransE, TransH, TransR.
- **Compositional Based Embeddings:** RESCAL, HoLE.
- **Neural Network Based Embeddings:** SLM,NTN,Conv2d.

These are described in detail in chapter 3.

1.3 Question Answering

Question Answering is a Natural Language Processing task in which human/user asks questions in natural language and expects a relevant answer from the algorithm. There

can be a different types of questions and based on that approaches may vary.

Different types of questions(Reddy et al. 2017) are :-

- **Factoid Type Questions :** These are fact-based questions. Usually factoid type questions start with 'wh' words. For example what is the capital of India? The answers are generally named entities.
- **List Type Questions :** These are the questions in which a list is required to answer them for example List the name of 10 comedy movies? The answer to this question will be a list of the named entities. Another example can be: List the steps to write a good report? The answer to this will not be a list of the named entities instead it will contain the list of statements in some order.
- **Confirmation Questions [yes or no] :** These are the questions in which the answer is a boolean (either yes or no). An example of confirmation-type questions is: Does the sun rises in the east? The answer is simply yes or no.
- **Non Factoid Type Questions :** These are open-ended questions that require complex answers to answer them. These can be opinions, descriptions, explanations. An example of non-factoid questions is How to read research papers? The answer to this question will contain some opinion and the answer will be descriptive.
- **Hypothetical Questions :** These are the type of questions that are related to hypothetical events i.e. the events which are not real. An example of hypothetical questions is what would you do if you are the Prime Minister of India?

1.4 Organization of Report

The rest of the report is organized as follows :-

- **Chapter 2 - Knowledge Graphs :** This chapter contains description of Knowledge Graphs such as WordNet (Section 2.1), ConceptNet (Section 2.3), BableNet (Section refsec$_b abelnet$), $Yago(Section 2.4), and Freebase(Section 2.5)$.
- **Chapter 3 - Knowledge Graph Embeddings :** This chapter contains the detailed description of knowledge graph embeddings and techniques used to generate these embeddings. This chapter is divided into further sections :-
  - In section 3.2 Translation based models are described such as TransE (3.2.1), TransH (3.2.2), TransR (3.2.3).
  - In section 3.3 Compositional based models are described such as RESCAL (3.3.1) ,HoLE (3.3.2).
  - In section 3.4 Neural Network based models are descibed such as SLM (3.4.1), NTN (3.4.2), Conv2d (3.4.3) etc.
  - In section 3.5 Open Knowledge Graph embeddings (Care 3.5.2) and Canonicalizing of Open Knowledge Bases using Embeddings and Side Information are discussed (3.5.1).
- **Chapter 4 Application of Knowledge Graph Embedding : Question Answering :**

  - In section 4.2 of this chapter knowledge graph embedding Question Answering is discussed which is an important application of knowledge graph embedding.
  - In section 4.4 automatic evaluation of question answering system is discussed. Automatic evaluation of question answering systems is hard so two transformer-based methods are described in this section.

- **Chapter 5 Question Answering using Deep Learning :** In this chapter first question answering is described in section 5.1. Some common datasets which can be used for training and evaluation is described in section 5.2. Two main deep learning based approached are described in section 5.6 and section 5.11.
- **Chapter 6 Joint IIT LG Project (Question Answering Using Knowledge Graph :** This chapter contains information about the joint IIT LG project. First problem statement is described in section 6.1. Almost every big project have pipeline, its described in section 6.2, Question Classifier is described in section 6.3, pre-trained QA systems is described in section 6.4. Natural language generation module is described in section 6.5, summarization module is described in section 6.6, UI is described in section 6.7.
- **Chapter 7 Summary , Conclusion and Future Work :** The last chapter contains the summary of the report (Section 7.1), conclusion (Section 7.2), and the description of future work (Section 7.3).

Chapter 2

Knowledge Graphs

This chapter describes various knowledge graphs and knowledge bases. Knowledge bases are the repository of the data (which can be structured or non-structured), Knowledge graphs represent knowledge with the help of graph (Structured). For all knowledge graphs, the introduction is given firstly and then various tools are described using which we can access the knowledge base. WordNet, which is a lexical database is described in section 2.1. Subsection 2.1.1 describes tools to use WordNet. BabelNet which is a multilingual semantic network is described in section 2.2. One of the important algorithms in the construction of BabelNet is the mapping algorithm which is described in subsection 2.2.1. Subsection 2.2.2 contains tools using which we can access BabelNet. ConceptNet which is a semantic network based on the OMCS database is described in Section 2.3, subsection 2.3.1 gives a description of some tools using which we can access ConceptNet. Similarly, a description of Yago is given in section 2.4, subsection 2.4.1. Freebase is described in section 2.5, tools for using freebase is described in section 2.5.1 .The chapter is summarized in section 2.6

2.1 WordNet

Wordnet is a lexical database (Miller et al. 1995) which is specially designed for implementing NLP tasks. The words in WordNet have syntactic categories. It contains open class syntactic categories that is *noun, verb, adjective* and *adverb*. Wordnet does not contain closed class categories such as *prepositions, pronouns, determiners*. The words which have similar meaning are grouped into synsets.

Every word in the dictionary has at least one sense. The words which have more than one sense are known as polysemous words and words with only one sense are known as monosemous words. WordNet has semantic relations between words. Some examples of semantic relations are *synonymy*, *antonymy*, *hyponymy*, *troponymy*, *entailment*. These are described below.

- **Synonymy:** It is symmetric relation between words which have similar meaning. In WordNet sense is defined by the set of synonyms which are called synsets. Because of this reason synonymy is considered as basic relation of wordnet for example magnify and expand.
- **Antonymy:** It is also symmetric relation between words which have the opposite

meaning, for example, humble and arrogant.

- **Hyponymy / Hypernymy:** It is a relation between two words in which one word(hyponym) has more specific semantic field than the other general word(hypernym) for example a dog is *hyponym* of an animal.
- **Meronymy / Holonymy:** It is a relation between two words in which one word(Meronym) is a part of the other word(Holonym) for example leaf is a *meronym* of tree.
- **Troponymy:** Troponymy is similar to hyponymy. Hyponymy is a relation between two words which are nouns. Troponymy is a relation between two words which are verbs for example walk is *troponym* of stroll.
- **Entailment:** It is also a relation between verbs in which one word entails the other word.

All the syntactic categories and semantic relations described above are summarized in the figure below 2.1 (Miller et al. 1995).

Semantic Relation	Syntactic Category	Examples
Synonymy (similar)	N, V, Aj, Av	pipe, tube rise, ascend sad, unhappy rapidly, speedily
Antonymy (opposite)	Aj, Av, (N, V)	wet, dry powerful, powerless friendly, unfriendly rapidly, slowly
Hyponymy (subordinate)	N	sugar maple, maple maple, tree tree, plant
Meronymy (part)	N	brim, hat gin, martini ship, fleet
Troponomy (manner)	V	march, walk whisper, speak
Entailment	V	drive, ride divorce, marry
Note: ***N*** = *Nouns* ***Aj*** = *Adjectives* ***V*** = *Verbs* ***Av*** = *Adverbs*		

Figure 2.1: Syntatic Categories And Semantic Relations in WordNet

2.1.1 Tools for using WordNet

In this section, firstly small introduction is given on how to use wordnet in python and then a small introduction about the web interface provided by wordnet so that it can be used online.

We can use wordnet in python using the nltk library (It is a very popular library for NLP). First, we need to import wordnet from the nltk library, and then we can use it for many things. Below figure 2.2 shows an example for using wordnet for finding synset of the word "dog".

```
[8]  from nltk.corpus import wordnet

[15] synset_dog = wordnet.synsets("dog")

[16] synset_dog

     [Synset('dog.n.01'),
      Synset('frump.n.01'),
      Synset('dog.n.03'),
      Synset('cad.n.01'),
      Synset('frank.n.02'),
      Synset('pawl.n.01'),
      Synset('andiron.n.01'),
      Synset('chase.v.01')]
```

Figure 2.2: Synstes of the word "dog"

WordNet consists of more than 166,000 word and sense pairs in which around 17 percent of the words are polysemous(words with more than one sense) and around 40 percent have synonyms (Miller et al. 1995).

There is a web interface (`http://wordnetweb.princeton.edu/perl/webwn`)in which the user can enter the word, select the display option and get the desired result. Below figure 2.3 gives an example of a search for the word "dog". We can see that synsets which we got from the above code 2.2 match this search.

Figure 2.3: Search for the word "dog"

2.2 BabelNet

BabelNet is a very large **multilingual** semantic network (Navigli et al. 2010). BabelNet combines lexical knowledge from WordNet (Miller et al. 1995) and encyclopedic knowledge from the Wikipedia. The creation of BabelNet is done automatically whereas the creation of WordNet was done manually. Both methods have their pros and cons. Integration of new knowledge can be done easily in BabelNet as creation is done automatically. The maintenance cost of BabelNet is also low compared to WordNet. Whereas WordNet is more accurate than BabelNet as the construction of WordNet is done manually.

Knowledge of WordNet and Wikipedia complements each other. Wikipedia has knowledge in unstructured format and WordNet provides a structure to the knowledge. If we can merge both these knowledge sources then we can get very rich source of structured knowledge. This is what BabelNet does, BabelNet is also a graph so it will contain vertices and edges. Vertices are words and edges are the relationship between the words. The steps of automatic construction of BabelNet is described below:-

- Vertices and Edges are extracted from WordNet without any change.
- Wikipedia pages contain hyperlinks. Hyperlinks are extracted from the page and each hyperlink represents a new node in the knowledge graph. They are connected by *unspecified* relation.
- Knowledge obtained from WordNet and Wikipedia is merged in this step as there can be duplicate entries so a mapping algorithm is used to identify those entries and merge them appropriately.
- Mapping is done by first disambiguating the word. For this authors (Navigli et al. 2010), took hyperlinks, sense labels, categories from Wiki page and synonymy, hypernymy/hyponymy, sisterhood, and gloss from WordNet.
- After mapping, authors used human-generated translation provided by Wikipedia and Google translate API for converting words to different contexts to make BabelNet multilingual.

The mapping steps are described in detail in section 2.2.1.

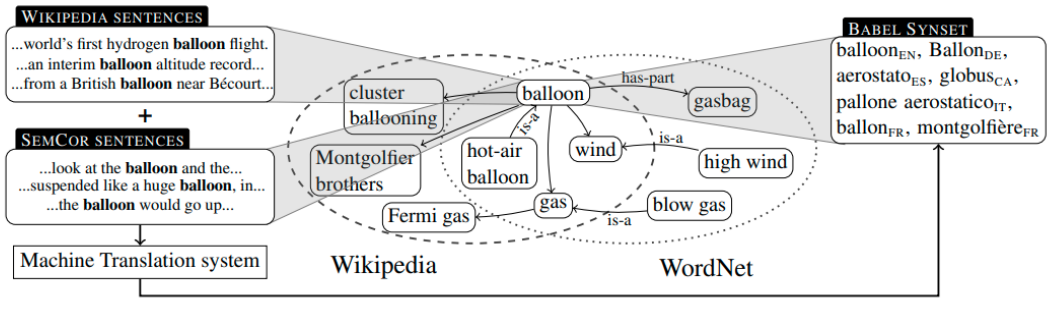


Figure 2.4: Example of BabelNet

‘ Figure 2.4 (Navigli et al. 2010). This figure shows an example of nodes and relations

in BabelNet. We can see relations of balloon (*haspart,is−a*) are taken from the wordnet. Hyperlinks such as *cluster ballooning, Fermi gas, etc.* are taken from Wikipedia. They are connected by an unspecified relation. SemCor is used by authors to make BabelNet multilingual. The figure also shows the Babel synset of the balloon which we get after performing the above operations.

2.2.1 Mapping Algorithm

WordNet and Wiki pages have common words and senses. Mapping them is an important task, to save memory and to preserve information. WordNet and Wiki pages complement each other so mapping is important to preserve information.

$$\mu(w) = \begin{cases} s \in Senses_{\text{WN}}(w) & \text{if a link can be established,} \\ \epsilon & \text{otherwise,} \end{cases}$$

$Senses_{WN}(w)$ is a set of senses for lemma of w and $\in$ denotes empty relation or unspecified relation. The steps to do mapping are:-

- Disambiguation context of Wiki page is obtained. Disambiguation context is a set of words obtained from *sense labels, links, categories* from Wiki page.
- Disambiguation context of Wordnet is obtained. Disambiguation context is a set of words obtained from *synonymy, hypernymy/hyponymy, sisterhood, gloss* from WordNet.
- The words which are monosemous are linked easily as they only have one sense.
- For polysemous words $p(s|w)$ is calculated.

$$\begin{aligned} \mu(w) = \underset{s \in Senses_{\text{WN}}(w)}{\text{argmax}} \; p(s|w) &= \underset{s}{\text{argmax}} \frac{p(s,w)}{p(w)} \\ &= \underset{s}{\text{argmax}} \, p(s,w) \end{aligned}$$

- $p(s|w)$ is defined as :-

$$p(s,w) = \frac{score(s,w)}{\sum\limits_{\substack{s' \in Senses_{\text{WN}}(w), \\ w' \in Senses_{\text{Wiki}}(w)}} score(s',w')}$$

Where $score(s,w) = |Ctx(s) \cap Ctx(w)| + 1$

2.2.2 Tools for using BabelNet

In this section, firstly introduction is given on how to use BabelNet in python and then a small introduction about the web interface provided by BabelNet so that it can be used online.

First step is to register at BabelNet (https://babelnet.org/register). We will get an API key, with the help of this key we can use services provided by BabelNet in python. Now download PyBabelNet package using pip (pip install pybabelnet). Below code snippet, 2.5 shows an example of using BabelNet in python.

```
[3] from py_babelnet.calls import BabelnetAPI

[4] api = BabelnetAPI('Your API Key')

[8] senses = api.get_senses(lemma = "dog", searchLang = "EN")
    senses
```

Figure 2.5: Example Code

We can also use BabelNet through the web https://babelnet.org/. Figure 2.6 shows an example of a search of the word "bottle".

Figure 2.6: Search of the word "bottle"

BabelNet is multilingual, in the above figure 2.6 Hindi translation of the same search is there (right side). We can see that some words are not translated in Hindi they are still in English. This is one of the issues with BabelNet.

2.3 ConceptNet

ConceptNet is a semantic network which is based on OMCS (Open Mind Common Sense) (Singh et al. 2002) database. Open Mind Common Sense contains some common sense facts for example *A doctor is a highly trained professional.* OMCS is just a collection of the facts in an unstructured format, which has very little use computationally. The structured format is required to make use of the common sense facts computationally. Here knowledge graph comes in, The knowledge graph or semantic network is one of the ways to make data in a structured format.

ConceptNet is a graph so it contains nodes and edges. Noun or noun phrase represent nodes and edges are statements of common sense. Figure 2.7 shows an example of ConceptNet.

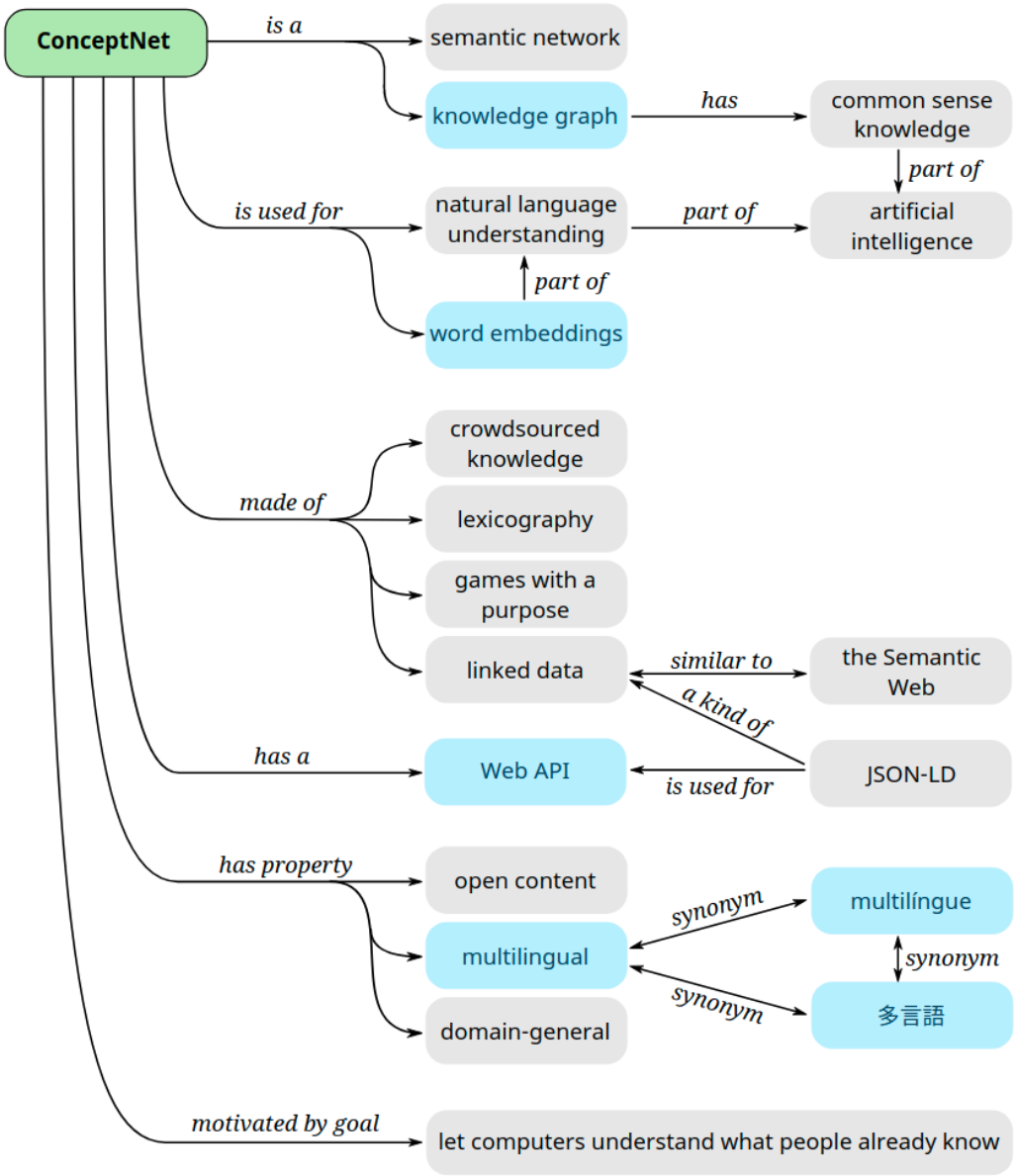


Figure 2.7: ConceptNet

The 2.7 (https://conceptnet.io/) shows an example of the ConceptNet . We can see the concepts which are nodes and assertions that are edges present in this graph.

2.3.1 Tools for using ConceptNet

In this section, firstly introduction is given on how to use ConceptNet in python and then a small introduction about the web interface provided by ConceptNet so that it can be used online.

We can use ConceptNet in python very easily. ConceptNet provides an API using that API we get a JSON object in return which contains all the information of the node which we requested using API. Figure 2.8 shows an example of using API to get all the information about the node which contains "dog".

```
[1] import requests

[2] response = requests.get('http://api.conceptnet.io/c/en/dog').json()
    response
```

Figure 2.8: Example Code

We can also use ConceptNet using web (https://conceptnet.io/). Figure 2.9 shows an example of a search of the word "dog". The figure only contains a small portion of the search result, for more information click on this https://conceptnet.io/c/en/dog

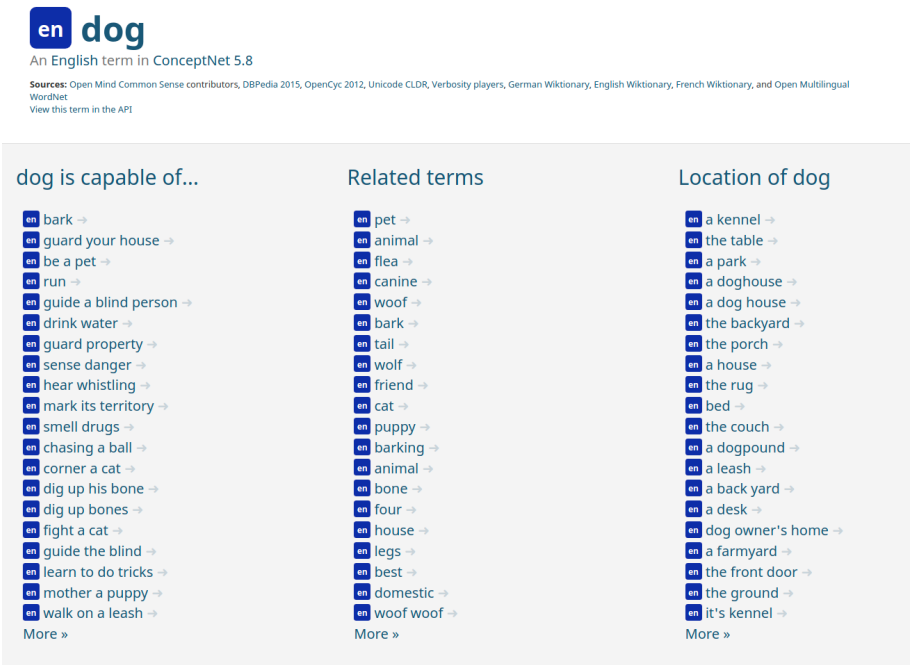


Figure 2.9: Search of the word "dog"

2.4 Yago

Yago is also a knowledge base that combines two great sources into one. It combines Wikidata and schema.org. Wikidata contains entities and is a great repository of the entities. Schema.org is an ontology of classes and relations but it does not contain entities. Yago combines these two resources to get information in structured format (Yago Site).

Yago is a knowledge graph so it will contain nodes and edges. Nodes are entities (Yago Site) such as (*people,countries,city*) and edges are relationship between these nodes. The latest version of Yago is 4 and it contains 50 million entities and 2 billion facts (Yago Site). Yago also has logical constraints. These constraints can be applied to entities as well as relations for example we can define a simple rule on birthPlace (Yago Site), one entity should be a person, and second, should be a place. These constraints helped in reducing incorrect data and also helped in reasoning. Domain and Range can also be specified as a logical constraint.

Data stored in Yago is of RDF format that is in the triplet (h, r, t) format, where h is the head entity. r is predicate/relation and t is tail entity. As Yago is a directed graph therefore we have head and tail entity both.

Entities in Yago are taken from Wikidata and relations from schema.org. They mapped relations from Wikidata to schema.org and discarded those relations which cannot be mapped.

There is also taxonomy in Yago which is taken from schema.org and BioSchemas. An example of the schema is shown in figure 2.10 (`https://yago-knowledge.org/graph/schema:Person`).

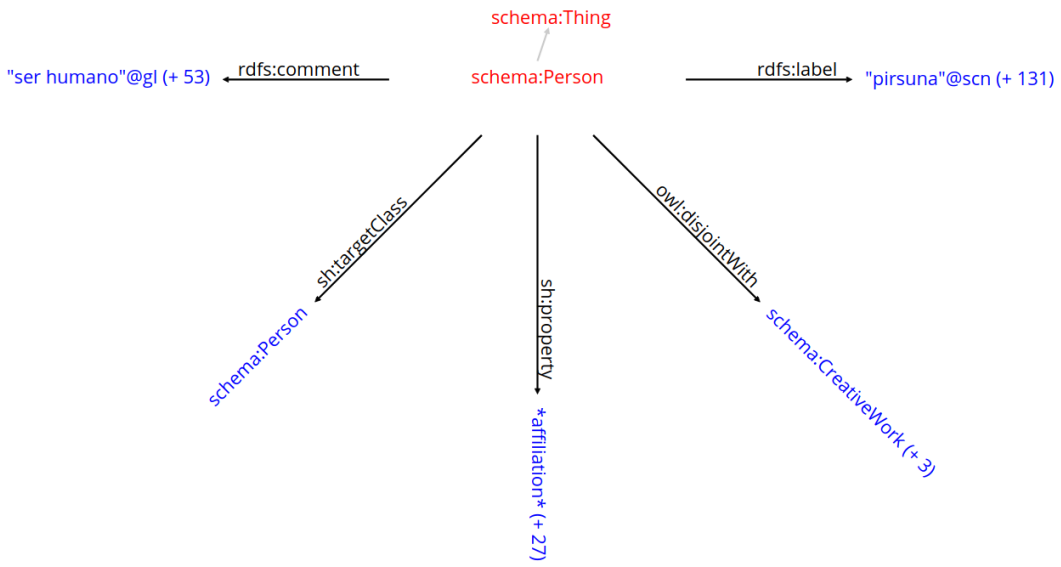


Figure 2.10: Schema:Person

2.4.1 Tools for using Yago

In this section, firstly introduction is given on how to use Yago using SPARQL query and then a small introduction about the web interface provided by Yago so that it can be used

online.

We can access Yago using SPARQL API. The endpoint should be https://yago-knowledge.org/sparql/query. We can also query using the web interface. A screenshot of the web interface provided by Yago using which we can give query and retrieve result is given in figure 2.11(https://yago-knowledge.org/sparql).

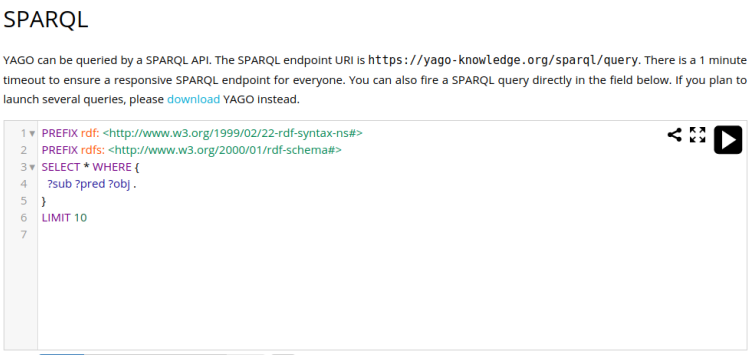


Figure 2.11: SPARQL Query Web Interface

We can also access Yago through an online interface (https://yago-knowledge.org/graph/Elvis_Presley). Figure 2.12 shows an example of such search.

Figure 2.12: Example of the search for the word "Dog"

2.5 Freebase

Freebase was a large knowledge base consisting of lots of entities and relationships. Google discontinued the freebase on 2 May 2016 (wikipedia). Freebase used Graphd (blog post by Scott Meyer) as the database which was created by them. Graphd is log-structured (append-only) database which means nothing is ever deleted from the database. Time stamp is used to distinguish between the new data and old data. We can even access the old data using the time stamp.

Figure 2.13 (Wang et al. 2014) shows statistics of some data sets. WN stands for WordNet. FB stands for Freebase. There are different sizes of Freebase dataset available. FB15k dataset have 14,951 entities and 1,345 relations. This data set is used when we have to experiment on how our algorithm is performing on large number of relations. FB13 contains only 13 relations and 75,043 entities. This data set is used when we have to experiment on how our algorithm is performing on small number of relations. FB5M contains 1,192 relations and 5,385,322 entities. This data set is used when we have to experiment on how much is our algorithm scalable.

Dataset	#R	#E	#Trip. (Train / Valid / Test)		
WN18	18	40,943	141,442	5,000	5,000
FB15k	1,345	14,951	483,142	50,000	59,071
WN11	11	38,696	112,581	2,609	10,544
FB13	13	75,043	316,232	5,908	23,733
FB5M	1,192	5,385,322	19,193,556	50,000	59,071

Figure 2.13: Freebase Data Statistics

2.5.1 Tools for using Freebase

Freebase API was discontinued by Google in 2016. There is data dump provided by google using which we can use Freebase (Data Dump). We can visit this site and download the last dumped data, which is approximately 250GB (uncompressed). Figure 2.14 (Data Dump) shows how we can download data dump of freebase.

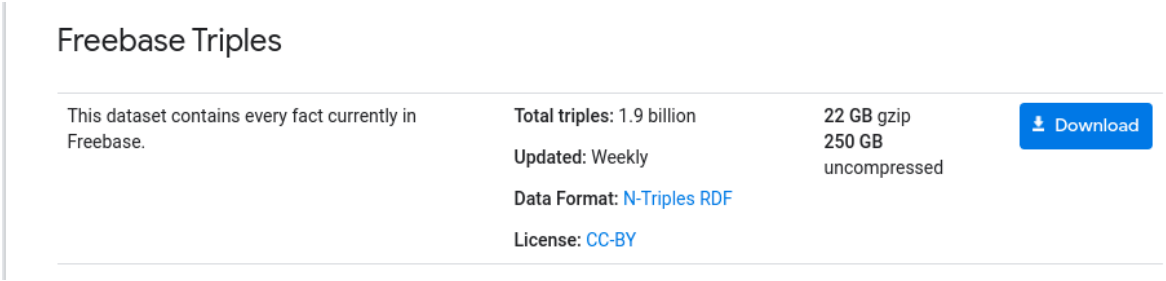


Figure 2.14: Data Dump of Freebase

As you can see in above figure 2.14 the format of data is RDF (<subject> <predicate> <object>) so querying on data is easy and similar to Yago.

2.6 Summary

We discussed various knowledge graphs which we can be used in NLP applications. We also discussed various tools through which we can apply the knowledge graph in our applications. We first discussed WordNet, which is a lexical database. We discussed various relationship that are between the words in the WordNet (*synonymy*, *antonymy*, *hyponymy*, *troponymy*, *entailment*). We then discussed the tools by which we can access the WordNet (nltk library and web). We discussed BabelNet, which is a very large multilingual semantix network. It is constructed using the data from both WordNet as well as Wikipages. There is a mapping algorithm which matches words from the wikipedia to the sense in the WordNet. Similiar to WordNet we then discussed the tools for using the BabelNet (using the Babelnet API and the Web). ConceptNet is also a semantic network which is based on the Open Mind Common Sense database, which contains facts like *A doctor is highly trained professional.* ConceptNet converts such facts which are in unstructed format to the semantic network which is in structured format. Similar to BabelNet we then discussed the API to use ConceptNet and the web interface to access the ConceptNet. Fourth knowledge base which we discussed was Yago, which combines two sources into one (Wikidata and Schema.org (and its siblings)). We also discussed some tools to access the Yago (both using SparQL and the web interface). Last knowledge base we discussed was Freebase which is very large knowledge base, but google discontinued the freebase on 2 May 2016. We can still download the dump of 2016 and use the freebase in our applications.

Chapter 3

Knowledge Graph Embeddings

Embeddings plays the crucial part in almost all NLP applications. This chapter contains a detailed description about the Knowledge Graph Embeddings. Section 3.1 gives the introduction, motivation, intuition about the knowledge graph embeddings. After the motivation, various knowledge graph embeddings are discussed, section 3.2 describes the important translation based embeddings like TransE (Section 3.2.1), TransH (Section 3.2.2), and TransR (Section 3.2.3). After discussing the translation based model we will discuss the models which uses the compositional operator (o) (Section 3.3), RESCAL uses the tensor product as the compositional operator which is discussed in detail in section 3.3.1. HoLE uses circular correlation as the compositional operator and it is discussed in section 3.3.2. Third kind of models discussed here are the neural based models, which are discussed in section 3.4, SLM (single layer model) are discussed in section 3.4.1. NTN (neural tensor product) are discussed at section 3.4.2. Two convolution based models are discussed in this chapter, Conv2d/ConvE is discussed in section 3.4.3. ConvKB is second convolution neural network based knowledge graph embeddings, which are discussed in section 3.4.4. Apart from these, embeddings of OpenKG (which are constructed using OpenIE techniques) are discussed in section 3.5. The knowledge graph constructed from the OpenIE methods have redundant nodes (words having same semantic meaning but are represented by different nodes), finding the redundant nodes is extremely important task. CESI algorithm for canonicalizing is discussed in section 3.5.1. We can use the same algorithms for generating embeddings of the OpenKG but these algorithms are not efficient due to some challenges in OpenKG. These challenges along with the algorithm for generating embeddings for OpenKG are discussed in section 3.5.2. The chapter is summarized in section 3.6

3.1 Introduction

Embeddings are the vector representation of things (words, images, videos, etc.) in the low dimension space. We try to capture meaning/similarity of things using these vector representations. Machine cannot understand things (words, images, videos, etc.) as we humans do, these vector representations are extremely important. These representation helps our machine learning algorithm to understand the meaning or difference in meanings of things (words, images, videos, etc.).

Word embeddings are very popular in NLP. Almost all applications in NLP uses word em-

beddings. Word embeddings are based on distributional hypothesis which suggests that if two words are semantically similar then they should be more distributionally similar ("you shall know a word by the company it keeps" - Firth). Similar words have similar representations for example dog and cat are similar, table and lion are not similar. This type of semantic properties are captured by the word embeddings.

Knowledge Graph embeddings are embeddings (vector representation) of entities and relations present in knowledge graph such that the structural properties of knowledge graph are still preserved. The intuition of knowledge graph embeddings and word embeddings is very similar. Similar entities(nodes) should have similar representation. Similar entities are those which have similar neighbors (which are connected by similar relations).

In Figure 3.1 (Open HPI KG Course) Stephen Hawking and Albert Einstein are similar nodes as they are connected with similar neighbors (Physicist and Person), thus they will have similar vector representation.

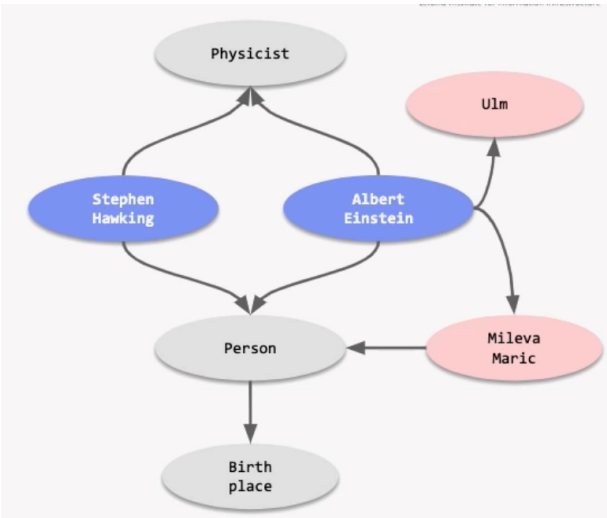


Figure 3.1: Intution Behind Knowledge Graph Embedding

Various application in NLP can be benefited from this knowledge graph embeddings for example Question Answering. Knowledge graph embedding captures the structure of the knowledge graph which can be of benefit in question answering task. Suppose we ask a question "When was dhoni born?" from a knowledge graph which have dhoni as an entity and born as a relation. We have representation of dhoni and born from knowledge graph embeddings and knowledge graph embeddings captures this structure knowledge so we can easily use this knowledge graph embeddings to get the answer "7 July 1981". Google's search engine uses the knowledge graph example of the same search is given in figure 3.2.

In the next section 3.2 we will discuss various algorithms for generating knowledge graph embedding. They are broadly divided into three categories: Translation-Based models which models relations (predicate) as a translation vector. Compositional-Based model which uses different compositional operators between entities and relations. Neural Network-Based models which uses neural network to model the relationship between the entities and relations.

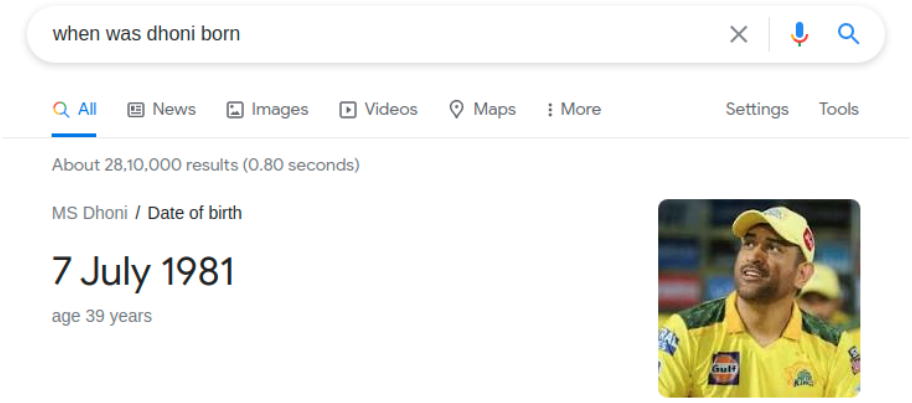


Figure 3.2: Google Search for the Question "When was dhoni born"

3.2 Translation-Based Models

We discussed about word embeddings in the above section. Word embeddings have a interesting translation invariance phenomena (Dai et al. 2020). Translation invariance can be seen in this famous example $\overrightarrow{King} - \overrightarrow{Queen} = \overrightarrow{Man} - \overrightarrow{Woman}$. Similar idea is applied to knowledge graph embedding. The algorithms which applies this similar idea comes under Translation-Based Models category. High level idea is that we assume relation as a translation vector which when applied to some entity(or after some operation on entity) we get another entity.

3.2.1 TransE

TransE is the first translation-based embedding proposed (Bordes et al. 2013). It assumes relation as a translation vector between head entity and tail entity. If we add relation vector with head entity we get a vector for tail entity.

$$\overrightarrow{h} + \overrightarrow{l} \approx \overrightarrow{t}$$

$\overrightarrow{h}$ denotes the head entity, $\overrightarrow{l}$ denote the predicate/relation, $\overrightarrow{t}$ denote the tail entity. This is the main equation for TransE model. Algorithm tries to make this relation possible for every triplet in the Knowledge Graph.

Training

We need a loss function to train our model. Margin based loss is used for training TransE in which entity embeddings and relation embeddings serves as a parameter to the model. The loss function is :

$$\mathscr{L} = \sum_{(h,l,t)\in S} \sum_{(h',l,t')\in S'_{(h,l,t)}} [\gamma + d(\mathbf{h}+\mathbf{l},\mathbf{t}) - d(\mathbf{h'}+\mathbf{l},\mathbf{t'})]_+$$

$d(\mathbf{h}+\mathbf{l},\mathbf{t})$ denotes the dissimilarity between $\overrightarrow{h} + \overrightarrow{l}$ and $\overrightarrow{t}$ which is difference between the vectors in this case. S denotes the triplet in the knowledge graph. S‘ denotes the corrupted triplet i.e the triplets which are not present in the knowledge graph. There is

a problem we only have set S, we don't have set S‘. How do we create this set S‘ ? we create this set by randomly replacing head or tail entity in S by some random entity, in this way we have set of corrupted triplets.

$$S' = \{(h', l, t)|h' \in E\} \cup \{(h, l, t')|t' \in E\}$$

. The training is carried out by using stochastic gradient descent with some additional constraints. L2 norm of the entities should be 1. This is important because loss can be reduced by increasing the norm of the entity vectors which we do not want. After the training is completed we will have nice property between entities and relations. Similar entities (which have similar neighbours and are connected by similar relationships) will have similar representation as they have to follow $\overrightarrow{h} + \overrightarrow{l} \approx \overrightarrow{t}$. Structural properties of the knowledge graphs is also preserved. The algorithm is described in the below figure 3.3 (Bordes et al. 2013).

Algorithm 1 Learning TransE

input Training set $S = \{(h, \ell, t)\}$, entities and rel. sets E and L, margin γ, embeddings dim. k.

initialize $\boldsymbol{\ell} \leftarrow$ uniform$(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each $\ell \in L$

  $\boldsymbol{\ell} \leftarrow \boldsymbol{\ell} / \|\boldsymbol{\ell}\|$ for each $\ell \in L$

  $\mathbf{e} \leftarrow$ uniform$(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each entity $e \in E$

loop

  $\mathbf{e} \leftarrow \mathbf{e} / \|\mathbf{e}\|$ for each entity $e \in E$

  $S_{batch} \leftarrow$sample(S, b) // sample a minibatch of size b

  $T_{batch} \leftarrow \emptyset$ // initialize the set of pairs of triplets

  for $(h, \ell, t) \in S_{batch}$ **do**

    $(h', \ell, t') \leftarrow$sample$(S'_{(h,\ell,t)})$ // sample a corrupted triplet

    $T_{batch} \leftarrow T_{batch} \cup \{((h, \ell, t), (h', \ell, t'))\}$

  end for

  Update embeddings w.r.t. $\sum_{((h,\ell,t),(h',\ell,t')) \in T_{batch}} \nabla \left[\gamma + d(\boldsymbol{h} + \boldsymbol{\ell}, \boldsymbol{t}) - d(\boldsymbol{h'} + \boldsymbol{\ell}, \boldsymbol{t'})\right]_+$

end loop

Figure 3.3: Algorithm TransE

Datasets used in Experiments

Datasets used by authors for evaluation are WordNet and Freebase. WordNet have less number of relations than freebase. Figure 3.4 (Bordes et al. 2013) shows the statistics of the datasets used in experiments. WN denotes WordNet and FB denotes Freebase

DATA SET	WN	FB15K	FB1M
ENTITIES	40,943	14,951	1×10^6
RELATIONSHIPS	18	1,345	23,382
TRAIN. EX.	141,442	483,142	17.5×10^6
VALID EX.	5,000	50,000	50,000
TEST EX.	5,000	59,071	177,404

Figure 3.4: Datasets (WordNet and Freebase)

WordNet (WN) have 18 relations which is small in comparison with Freebase (FB). Two version of Freebase datasets are used for experiments. FB15K is subset of FB1M containing 1,345 relations and 14,951 (nearly 15K) entities. FB1M contain 1M entities and 23,382 relations. The division between train, validation and test set is given in the figure 3.4 (Bordes et al. 2013).

Evaluation Results on Link Prediction Task

Evaluation of the algorithm (Bordes et al. 2013) is done on link prediction task by the authors. Link prediction is a task of predicting head/tail entity given the predicate and head/tail entity. First head is removed and replaced by each of the entity in knowledge graph. Ranking is done based on the dissimilarity score. The rank of correct triplet is stored. Similar procedure is done for the tail entity. Mean rank and hits@10 is reported for the link prediction task. Mean rank is the mean of the predicted rank and hits@10 is the proportion of the correct result reported in the top 10 rank.

This method have a problem. What if the corrupted triplet is the correct triplet, as we are replacing head/tail entity by all the entities in the knowledge graph, its possible that we get is correct triplet (after replacing. As there can be many to many/many to one/one to many relations). If algorithm ranks such triplet above the test triplet (which we are considering only correct triplet) then it will count as error but its not an error. To eliminate this problem authors proposed *filter* setting in which all the triplet are removed from corrupted triplet if they belong to training set, validation set or test set, in this way the error due to this problem is minimized.

Dataset	WN				FB15k				FB1M	
Metric	Mean Rank		Hits@10 (%)		Mean Rank		Hits@10 (%)		Mean Rank	Hits@10 (%)
Eval. setting	*Raw*	*Filt.*	*Raw*	*Filt.*	*Raw*	*Filt.*	*Raw*	*Filt.*	*Raw*	*Raw*
Unstructured [2]	315	304	35.3	38.2	1,074	979	4.5	6.3	15,139	2.9
RESCAL [11]	1,180	1,163	37.2	52.8	828	683	28.4	44.1	-	-
SE [3]	1,011	985	68.5	80.5	273	162	28.8	39.8	22,044	17.5
SME(linear) [2]	545	533	65.1	74.1	274	154	30.7	40.8	-	-
SME(bilinear) [2]	526	509	54.7	61.3	284	158	31.3	41.3	-	-
LFM [6]	469	456	71.4	81.6	283	164	26.0	33.1	-	-
TransE	**263**	**251**	**75.4**	**89.2**	**243**	**125**	**34.9**	**47.1**	**14,615**	**34.0**

Figure 3.5: Result on Link Prediction Task

The above figure 3.5 (Bordes et al. 2013) contains result for the link prediction task. Two metric which authors used were mean rank(lower the better) and hits@10(higher the better). The results are reported in raw(without filtering) setting as well as filter(described above) setting. Unstructured is the baseline considered by the authors. In unstructured $\overrightarrow{l}$ is assumed to be zero. The entities which are related to each other will be clustered together in unstructured algorithm. We can see from the result that TransE performed better than state of the art algorithms.

All the algorithm in the figure 3.5(Bordes et al. 2013) except TransE, Unstructured requires lot of parameters (in the order of square). TransE only requires $O(n_e k + n_r k)$ parameters, where n_e is number of entity and n_r is number of relations and k is the dimension of the entity and relation embeddings. Why is this important that TransE have less number of parameters compared to other algorithms? The answer is that less number of parameters make TransE scalable, see coloumn FB1M in figure 3.5(Bordes et

al. 2013) only results for Unstructured, TransE and SE are reported. The reason is other algorithms are not scalable which makes them impractical for large knowledge graph.

TransE is not only the better algorithm in terms of lower mean rank and higher hit@10 rank. It is also scalable, it takes less time to converge which makes it practical to use for large Knowledge Graph also.

3.2.2 TransH

TransH (Knowledge Graph Embedding by Translating on Hyperplanes (Wang et al. 2014) is the second translation based embedding which we will discuss. Before discussing new embedding we should first look into what are the problems with TransE algorithm and why we need another translation based model.

Consider the ideal case in which there is no error after training in TransE i.e $\mathbf{h}+\mathbf{r}-\mathbf{t}=0$ (r and l represents same thing(relation/predicate)):

- If $(h,r,t) \in S$ and $(t,r,h) \in S$ i.e both triplets are in knowledge graph. This means r is reflexive relation. In case of TransE (ideal case) $\mathbf{r}=0$ and $\mathbf{h}=\mathbf{t}$. Which should not be the case there can be reflexive relation in knowledge graph example brother relation which is reflexive.
- If $\forall i \in \{0,...,m\}$, $(h_i,r,t) \in S$ i.e r is many to one relation. In case of TransE (ideal case), all the head entities will be same ($\mathbf{h_0}=...=\mathbf{h_m}$). Which should not be the case all the head entities should not have same representation as they are different entities.
- If $\forall i \in \{0,...,m\}$, $(h,r,t_i) \in S$ i.e r is one to many relation. In case of TransE (ideal case), all the tail entities will be same ($\mathbf{t_0}=...=\mathbf{t_m}$). Which should not be the case all the tail entities should not have same representation as they are different entities.

These cases represents the problem/issues with TransE embedding. Figure 3.6 (Bordes et al. 2013) shows the results of TransE based on category of relationship. We can see that in many to one case (predicting head) the hits@10 result suffer (18.2). In one to many case (predicting tail) the hits@10 result suffer (19.7). Both these results supports above hypothesis that TransE is unable to capture many to one (predicting head) and one to many (predicting tail) relationships.

TASK	PREDICTING *head*				PREDICTING *tail*			
REL. CATEGORY	1-TO-1	1-TO-M.	M.-TO-1	M.-TO-M.	1-TO-1	1-TO-M.	M.-TO-1	M.-TO-M.
Unstructured [2]	34.5	2.5	6.1	6.6	34.3	4.2	1.9	6.6
SE [3]	35.6	62.6	17.2	37.5	34.9	14.6	68.3	41.3
SME(LINEAR) [2]	35.1	53.7	19.0	40.3	32.7	14.9	61.6	43.3
SME(BILINEAR) [2]	30.9	**69.6**	**19.9**	38.6	28.2	13.1	**76.0**	41.8
TransE	**43.7**	65.7	18.2	**47.2**	**43.7**	**19.7**	66.7	**50.0**

Figure 3.6: Results by Category of Relationship

To solve this problem TransH (Wang et al. 2014) proposes relation specific hyperplanes. Each relation have a translation vector d_r (This is similar to TransE) and a relation specific hyperplane w_r (which a normal vector of the plane).

Figure 3.7 (Wang et al. 2014) gives an example of how head, relation/predicate, tail entity are treated in transH. the relation specific plane is represented by dashed plane. $\mathbf{h}$ (head entity) and $\mathbf{t}$ (tail entity) are first projected in the relation specific plane. $\mathbf{h}_\perp$ denotes head entity projected in relation specific plane, $\mathbf{t}_\perp$ denotes tail entity projected in relation specific plane. Assume (h, r, t) is the golden triplet present in our dataset. The scoring function for transH will be $||\mathbf{h}_\perp + \mathbf{d}_r - \mathbf{t}_\perp||_2^2$. Our goal is to make this scoring function as small as possible, in other words we want that the vector addition between $\mathbf{h}_\perp$ and $\mathbf{d}_r$ should be equal to $\mathbf{t}_\perp$.

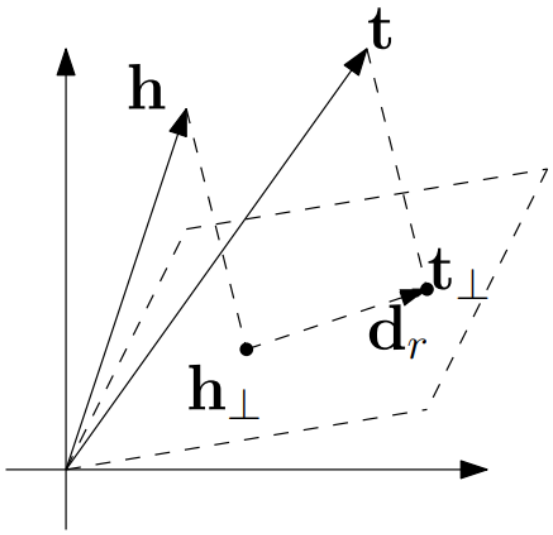


Figure 3.7: TransH

Now we can use simple vector algebra to get the value of $\mathbf{h}_\perp$ and $\mathbf{t}_\perp$.

$||\mathbf{w}_r|| = 1$ *(Restriction/Assumption)*
$\mathbf{w}_r^T\mathbf{h}$ *(Magnitude of projection of* $\mathbf{h}$ *in the direction of* $\mathbf{w}_r(h\cos\theta)$*)*
$\mathbf{w}_r^T\mathbf{h}\mathbf{w}_r$ *(Projection of* $\mathbf{h}$ *in the direction of* $\mathbf{w}_r$*(dashed line in figure 3.7))*
$\mathbf{h}_\perp + \mathbf{w}_r^T\mathbf{h}\mathbf{w}_r = \mathbf{h}$ *(Triangle law for vector addition)*
$\mathbf{h}_\perp = \mathbf{h} - \mathbf{w}_r^T\mathbf{h}\mathbf{w}_r$
$\mathbf{t}_\perp = \mathbf{t} - \mathbf{w}_r^T\mathbf{t}\mathbf{w}_r$ *(Similar to above steps)*

The score function becomes :

$$f_r(h,t) \;=\; ||(\mathbf{h} - \mathbf{w}_r^T\mathbf{h}\mathbf{w}_r) \;+\; d_r \;-\; (\mathbf{t} - \mathbf{w}_r^T\mathbf{t}\mathbf{w}_r)||_2^2$$

This score function should give low score to the correct triplet i.e. the triplet which belongs to our knowledge graph and high score to those triplets which does not belong to our knowledge graph. Loss function for training is designed keeping in mind these objectives (Wang et al. 2014).

Training

The parameters of the model are entities embeddings($\{\mathbf{e}_i\}_{i=1}^{|E|}$), the relations hyperplanes(normal vector)($\{\mathbf{w}_r\}_{r=1}^{|R|}$) and translation vector($\{\mathbf{d}_r\}_{r=1}^{|R|}$). We need a loss function for training our model. Similar to TransE, TransH also uses margin based loss function.

$$\mathscr{L} = \sum_{(h,l,t)\in S} \sum_{(h',l,t')\in S'_{(h,l,t)}} [\gamma + f_r(\mathbf{h},\mathbf{t}) - f_{r'}(\mathbf{h'},\mathbf{t'})]_+$$

f_r is the scoring function. S is a set of all the valid triplet. S' is a set of all the corrupted triplet. γ is the hyper-parameter. Similar to TransE, TransH also have some constrains(Wang et al. 2014).

$$\forall e \in E, \|\mathbf{e}\|_2 \leq 1, \text{//scale}$$
$$\forall r \in R, |\mathbf{w}_r^\top \mathbf{d}_r| / \|\mathbf{d}_r\|_2 \leq \epsilon, \text{//orthogonal}$$
$$\forall r \in R, \|\mathbf{w}_r\|_2 = 1, \text{//unit normal vector}$$

Instead of optimizing the loss function with hard constraints, authors optimize the loss function using soft constraints.

$$\mathscr{L} = \sum_{(h,l,t)\in S} \sum_{(h',l,t')\in S'_{(h,l,t)}} [\gamma + f_r(\mathbf{h},\mathbf{t}) - f_{r'}(\mathbf{h'},\mathbf{t'})]_+ + C\left\{\sum_{e\in E}\left[||\mathbf{e}||_2^2 - 1\right]_+ + \sum_{r\in R}\left[\frac{\left(\mathbf{w}_r^T\mathbf{d}_r\right)^2}{||\mathbf{d}_r||_2^2} - \epsilon^2\right]_+\right\}$$

C is the hyper-parameter weighing the importance of these soft constraints. Stochastic gradient descent (SGD) is used to minimize the loss.

Reducing False Negative Labels

Sharp readers might have noticed the problem with loss function of TransE. The problem is how TransE generate the negative examples (S'). TransE generates negative examples by first fixing head entity and changing tail entity and then fixing tail entity and changing head entity.

$$S' = \{(h', r, t)|h' \in E\} \cup \{(h, r, t')|t' \in E\}$$

There is a problem, relations can be many to many i.e one entity is related to more than one entities (example sibling relation). Its possible that $((h', r, t))$ is a valid triplet but we are considering it to be a corrupted triplet, which is wrong. It will affect the performance of the model.

The solution(Wang et al. 2014) to this problem is simple. We can create negative examples by replacing head entity in one to many relations. As there is only one correct head entity possible in one to many relations so any other entity in place of head entity will be a corrupted triplet. Similarly we replace tail entity in many to one relation for similar reasons.

How can we identify one to many and many to one relations in our knowledge graph? For identifying we use Bernoulli distribution.

For every relation r we define :

$$tph \quad (Average\ number\ of\ tail\ entity\ per\ head\ entity)$$
$$hpt \quad (Average\ number\ of\ head\ entity\ per\ tail\ entity)$$
$$\frac{tph}{tph + hpt} \quad (Bernoulli\ distribution\ parameter)$$

Now we perform sampling. With probability $\frac{tph}{tph+hpt}$ we replace head entity. Reason is simple, if tail per head entity is more than head per tail entity then its like one to many relation. Similarly with probability $\frac{hpt}{tph+hpt}$ we replace tail entity (similar to many to one relation).

Datasets used in Experiments

Datasets (3.8) used for experiments by authors(Wang et al. 2014) are almost same as TransE. WN18, FB15K are the same database used in experiments in TransE model. WN11 is the WordNet database with 11 realtions and 38,696 entities. FB13 contains only 13 relations which is used to evaluate performance of model in less number of relation setting. FB5M is a very large knowledge graph consisting of 1,192 relation and nearly 5M enities. This can be used to test scalability of the model.

Dataset	#R	#E	#Trip. (Train / Valid / Test)		
WN18	18	40,943	141,442	5,000	5,000
FB15k	1,345	14,951	483,142	50,000	59,071
WN11	11	38,696	112,581	2,609	10,544
FB13	13	75,043	316,232	5,908	23,733
FB5M	1,192	5,385,322	19,193,556	50,000	59,071

Figure 3.8: Datasets

Evaluation Results on Link Prediction Task

Link prediction Task is described in section 3.2.1. Results are shown in figure 3.9(Wang et al. 2014). Raw and Filter settings are same as TransE and is described in section 3.2.1. "Unif" denotes the traditional way of creating corrupted triplets. "Bern" denotes the way descibed in section 3.2.2. As usual lower Mean rank is better and higher Hits@10 rate is better.

Results are reported for WN18 and FB15K datasets. Unstructured performed better in WN18 because the number of relation in WN18 is very less. Overall TransH performed consistently better than all the other embeddings algorithms. TransE is runner up in terms of performance on both the datasets.

As you may recall TransH was proposed because of some of the problems of TransE. TransE does not perform good on one to many relation (predicting tail entity), many to one relations (predicting head entity). Figure 3.10(Wang et al. 2014) shows result on link prediction task performed on FB15K dataset category wise.

Dataset	WN18				FB15k			
Metric	MEAN		HITS@10		MEAN		HITS@10	
	Raw	Filt.	Raw	Filt.	Raw	Filt.	Raw	Filt.
Unstructured (Bordes et al. 2012)	315	304	35.3	38.2	1,074	979	4.5	6.3
RESCAL (Nickel, Tresp, and Kriegel 2011)	1,180	1,163	37.2	52.8	828	683	28.4	44.1
SE (Bordes et al. 2011)	1,011	985	68.5	80.5	273	162	28.8	39.8
SME (Linear) (Bordes et al. 2012)	545	533	65.1	74.1	274	154	30.7	40.8
SME (Bilinear) (Bordes et al. 2012)	526	509	54.7	61.3	284	158	31.3	41.3
LFM (Jenatton et al. 2012)	469	456	71.4	81.6	283	164	26.0	33.1
TransE (Bordes et al. 2013b)	263	**251**	75.4	**89.2**	243	125	34.9	47.1
TransH (unif.)	318	**303**	75.4	**86.7**	211	**84**	42.5	**58.5**
TransH (bern.)	400.8	388	73.0	82.3	212	**87**	45.7	**64.4**

Figure 3.9: Link Prediction Results

Task	Predicting left (HITS@10)				Predicting right (HITS@10)			
Relation Category	1-to-1	1-to-n	n-to-1	n-to-n	1-to-1	1-to-n	n-to-1	n-to-n
Unstructured (Bordes et al. 2012)	34.5	2.5	6.1	6.6	34.3	4.2	1.9	6.6
SE (Bordes et al. 2011)	35.6	62.6	17.2	37.5	34.9	14.6	68.3	41.3
SME (Linear) (Bordes et al. 2012)	35.1	53.7	19.0	40.3	32.7	14.9	61.6	43.3
SME (Bilinear) (Bordes et al. 2012)	30.9	69.6	19.9	38.6	28.2	13.1	76.0	41.8
TransE (Bordes et al. 2013b)	43.7	65.7	18.2	47.2	43.7	19.7	66.7	50.0
TransH (unif.)	**66.7**	**81.7**	**30.2**	**57.4**	**63.7**	**30.1**	**83.2**	**60.8**
TransH (bern.)	**66.8**	**87.6**	**28.7**	**64.5**	**65.5**	**39.8**	**83.3**	**67.2**

Figure 3.10: Link Prediction Results by Relation Category on FB15K

TransH performs much better in many to one (predicting head/left entity), one to many (predicting tail/right entity) compared to TransE and all other embedding algorithms. Performance of TransH also increased significantly in one to one, one to many (predicting head/left entity), many to one (predicting tail/right entity), many to many. Reason (Wang et al. 2014) is that embeddings have global effect i.e. better embeddings of some parts lead to better results on the whole (Wang et al. 2014).

Triplet Classification Task

Triplet classification task(Wang et al. 2014) is the binary classification task. Given a input triplet (h, r, t), task is to confirm whether the triplet belongs to the knowledge graph or not. Figure 3.11(Wang et al. 2014) shows the triplet classification result on WN11, FB13 and FB15K dataset. Statistics of the datasets is given in figure 3.8.

Accuracy is used as a metric in triplet classification task. Numbers shown in bracket are the time taken by algorithm to train. TransH perform best on the WN11 dataset. The result on FB15K are reported only for the model NTN, TransE and TransH reason in FB15K contains 1,345 relations. All other algorithms except TransE and TransH are not scalable so its not possible to train them in FB15K setting. TransH and TransE both performed good on FB15K dataset. The intresting thing to note here is TransE only takes 5 minute to train on FB15K which is really cool, by only training for 5 minutes we can get 87.3% accuracy. NTN takes approximately 40 hours to train on FB15K, achieves only 66.5% accuracy, which shows that NTN is not scalable. TransH takes 30 minutes to train which is more than TransE but significantly way less then NTN. The reason that

Dataset	WN11	FB13	FB15k
Distant Model	53.0	75.2	-
Hadamard Model	70.0	63.7	-
Single Layer Model	69.9	**85.3**	-
Bilinear Model	73.8	84.3	-
NTN	70.4	**87.1**	66.5 ($\approx 40h$)
TransE (unif.)	75.85	70.9	79.7 ($\approx 5m$)
TransE (bern.)	75.87	81.5	**87.3** ($\approx 5m$)
TransH (unif.)	**77.68**	76.5	80.2 ($\approx 30m$)
TransH (bern.)	**78.80**	83.3	**87.7** ($\approx 30m$)

Figure 3.11: Triplet Classification Result

NTN takes more time to train compared to TransE and TransH is that NTN have more number of trainable parameter compared to both TransE and TransH.

3.2.3 TransR and CTransR

TransR (Lin et al. 2015) is another translation based embedding.In TransH entities are first projected in the relation specific hyper-plane, then the translation is applied. In TransR first entities are projected in separate relational space, then the translation is applied. Relations and Entities do not share space unlike TransE and TransH since both relation and entity are different things. Every relation have a separate space and a translation vector. Figure 3.12(Lin et al. 2015) shows a simple illustration of TransR.

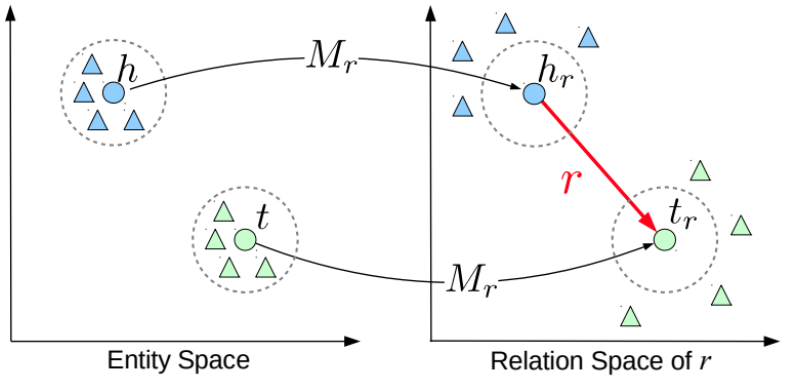


Figure 3.12: Illustration TransR

Head entity(h) is projected in the relation specific space with the help of matrix M_r. Similarly tail entity is projected onto relational specific plane. Translation is then applied on h_r to get t_r i.e. $h_r + r = t_r$.

$$\mathbf{h_r} = \mathbf{hM_r} \quad \text{(Projection Head entity to Relation space)}$$
$$\mathbf{t_r} = \mathbf{tM_r} \quad \text{(Projection Tail entity to Relation space)}$$
$$f_r(h,t) = \|\mathbf{h_r} + \mathbf{r} - \mathbf{t_r}\|_2^2 \quad \text{(Score Function)}$$

We want that the score function should give low score to the triplet which belongs to our knowledge graph and high score to those triplet which does not belong to our knowledge graph.

CTransR

TransE, TransH, TransR assume a single relational vector (translation vector). This is not the case a relation may have different representation depending on the type of entities it is dealing with. For example the relation "location_location_contains"(Lin et al. 2015) can have head entity as a continent and tail entity as a country, or head entity as a country and tail entity as a states. Depending on the type of head and tail entity relation can have different representation. Figure 3.13(Lin et al. 2015) shows an example the same. Asia is continent and Nepal is a country, Italy is a country and Milan Conservatory is an university. Same relation can have different types of head and tail entities.

	⟨Head, Tail⟩
1	⟨Africa, Congo⟩, ⟨Asia, Nepal⟩, ⟨Americas, Aruba⟩, ⟨Oceania, Federated States of Micronesia⟩
2	⟨United States of America, Kankakee⟩, ⟨England, Bury St Edmunds⟩, ⟨England, Darlington⟩, ⟨Italy, Perugia⟩
3	⟨Georgia, Chatham County⟩, ⟨Idaho, Boise⟩, ⟨Iowa, Polk County⟩, ⟨Missouri, Jackson County⟩, ⟨Nebraska, Cass County⟩
4	⟨Sweden, Lund University⟩, ⟨England, King's College at Cambridge⟩, ⟨Fresno, California State University at Fresno⟩, ⟨Italy, Milan Conservatory⟩

Figure 3.13: Example of Relation "location_location_contains"

The basic idea is, first for specific relation r, (h, t) pairs are grouped. Every entity pairs are represented by their offset $(\mathbf{h} - \mathbf{t})$. Clustering is performed using this offset for each pair ($\mathbf{h}$ and $\mathbf{t}$ are obtained using TransE). Now for each relation CTransR learns the projection matrix ($\mathbf{M_r}$) and for each cluster it learns a translation vector ($\mathbf{r_c}$). Scoring function for TransR is :

$$f_r(h,t) = \|\mathbf{h}_{r,c} + \mathbf{r}_c - \mathbf{t}_{r,c}\|_2^2 + \alpha\|\mathbf{r}_c - \mathbf{r}\|_2^2 \qquad \text{(Score Function)}$$

The second term $\|\mathbf{r}_c - \mathbf{r}\|_2^2$ ensures that the cluster-specif relation vector $\mathbf{r}_c$ is not too far away from orignal relation $\mathbf{r}$.

Training

We need a loss function for training (TransR and CTransR). Similar to TransE(Bordes et al. 2013) and TransH(Wang et al. 2014), TransR(Lin et al. 2015) and CTransR(Lin et al. 2015) uses margin based loss function.

$$\mathscr{L} = \sum_{(h,l,t)\in S} \sum_{(h',l,t')\in S'_{(h,l,t)}} [\gamma + f_r(\mathbf{h},\mathbf{t}) - f_{r'}(\mathbf{h'},\mathbf{t'})]_+$$

f_r is the scoring function. S is a set of all the valid triplet. $S^{'}$ is a set of all the corrupted triplet. SGD is used for training TransR and CTransR model using this loss function. where $\mathbf{M_r}$, $\mathbf{r}$, $\mathbf{h}$, $\mathbf{t}$ are the parameters of the model in case of TransR. $\mathbf{M_r}$, $\mathbf{r}_c$, $\mathbf{rh}$, $\mathbf{t}$ are parameter of the model in case for CTransR.

Evaluation Results on Link Prediction Task

Link prediction task is described in section 3.2.1 . Datasets used for this experiment are same as the datasets used in the case of TransH (see section 3.8). Figure 3.14(Lin et al. 2015) shows the result of the TransR and CTransR model on the link prediction task.

Data Sets	WN18				FB15K			
Metric	Mean Rank		Hits@10 (%)		Mean Rank		Hits@10 (%)	
	Raw	Filter	Raw	Filter	Raw	Filter	Raw	Filter
Unstructured (Bordes et al. 2012)	315	304	35.3	38.2	1,074	979	4.5	6.3
RESCAL (Nickel, Tresp, and Kriegel 2011)	1,180	1,163	37.2	52.8	828	683	28.4	44.1
SE (Bordes et al. 2011)	1,011	985	68.5	80.5	273	162	28.8	39.8
SME (linear) (Bordes et al. 2012)	545	533	65.1	74.1	274	154	30.7	40.8
SME (bilinear) (Bordes et al. 2012)	526	509	54.7	61.3	284	158	31.3	41.3
LFM (Jenatton et al. 2012)	469	456	71.4	81.6	283	164	26.0	33.1
TransE (Bordes et al. 2013)	263	251	75.4	89.2	243	125	34.9	47.1
TransH (unif) (Wang et al. 2014)	318	303	75.4	86.7	211	84	42.5	58.5
TransH (bern) (Wang et al. 2014)	401	388	73.0	82.3	212	87	45.7	64.4
TransR (unif)	232	219	78.3	91.7	226	78	43.8	65.5
TransR (bern)	238	225	**79.8**	92.0	**198**	77	48.2	68.7
CTransR (unif)	243	230	78.9	**92.3**	233	82	44	66.3
CTransR (bern)	**231**	**218**	79.4	**92.3**	199	**75**	**48.4**	**70.2**

Figure 3.14: Link Prediction Results

The results is reported in Raw and Filter settings both (see section 3.2.1). TransH, TransR and CTransR are trained on both unif and bern settings (see section 3.2.2) . We can clearly see that TransR and CTransR both outperform all the translation based techniques, having low Mean Rank and high Hits@10 (%). Figure 3.15(Lin et al. 2015) shows the result of link prediction task (relation category wise).

Tasks	Predicting Head(Hits@10)				Predicting Tail(Hits@10)			
Relation Category	1-to-1	1-to-N	N-to-1	N-to-N	1-to-1	1-to-N	N-to-1	N-to-N
Unstructured (Bordes et al. 2012)	34.5	2.5	6.1	6.6	34.3	4.2	1.9	6.6
SE (Bordes et al. 2011)	35.6	62.6	17.2	37.5	34.9	14.6	68.3	41.3
SME (linear) (Bordes et al. 2012)	35.1	53.7	19.0	40.3	32.7	14.9	61.6	43.3
SME (bilinear) (Bordes et al. 2012)	30.9	69.6	19.9	38.6	28.2	13.1	76.0	41.8
TransE (Bordes et al. 2013)	43.7	65.7	18.2	47.2	43.7	19.7	66.7	50.0
TransH (unif) (Wang et al. 2014)	66.7	81.7	30.2	57.4	63.7	30.1	83.2	60.8
TransH (bern) (Wang et al. 2014)	66.8	87.6	28.7	64.5	65.5	39.8	83.3	67.2
TransR (unif)	76.9	77.9	**38.1**	66.9	76.2	38.4	76.2	69.1
TransR (bern)	78.8	**89.2**	34.1	69.2	79.2	37.4	**90.4**	72.1
CTransR (unif)	78.6	77.8	36.4	68.0	77.4	37.8	78.0	70.3
CTransR (bern)	**81.5**	89.0	34.7	**71.2**	**80.8**	38.6	90.1	**73.8**

Figure 3.15: Link Prediction Results by Relation Category on FB15K

We can see from above results that TransR and CTransR are performing better in many to one (predicting head) case, one to many (predicting tail) case compared to all other

embeddings algorithms but the accuracy percentage are still low if we compare to other relation categories which is a major concern.

Triplet Classification Task

Triplet classification task is defined in section 3.2.2. Figure 3.16 (Lin et al. 2015) shows the result of triplet classification task. Metric shown in the figure is accuracy. TransR and CTransR performed best on WN11 and FB15K dataset. The result for some models are not shown in the figure due to scalability issue. Accuracy of TransR and CTransR are higher but TransE requires only 5 minute to train. TransE is much simpler model compared to TransR and CTransR. While TransR and CTransR give good results it lacks simplicity and requires much higher time for training. In translation models there is a trade-off between simplicity/time taken for training/scalability and performance.

Data Sets	WN11	FB13	FB15K
SE	53.0	75.2	-
SME (bilinear)	70.0	63.7	-
SLM	69.9	85.3	-
LFM	73.8	84.3	-
NTN	70.4	**87.1**	68.5
TransE (unif)	75.9	70.9	79.6
TransE (bern)	75.9	81.5	79.2
TransH (unif)	77.7	76.5	79.0
TransH (bern)	78.8	83.3	80.2
TransR (unif)	85.5	74.7	81.7
TransR (bern)	**85.9**	82.5	83.9
CTransR (bern)	85.7	-	**84.5**

Figure 3.16: Triplet Classification Result

3.3 Compositional-Based Models

Compositional-Based Models are the second type of embedding algorithms which we will discuss in this chapter. These type of models are very similar, they only differ in the compositional operator they use (Nickel et al. 2016).

Let us denote a triplet by $R_p(s, o)$ (subject, predicate, object). Some literature uses this notation while others uses (h, r, t). In this section we will use $R_p(s, o)$ notation. For every relation we define a characterstic function $\Phi_p : \mathscr{E} X \mathscr{E} \implies \{\pm 1\}$. $\mathscr{E}$ denotes the Entity .Φ_p is function which tells us whether the triplet (s, p, o) belongs to knowledge graph or not. For every pair of entity this characteristic function is defined. Now we can generalized the learning model by this formula(Nickel et al. 2016).

$$\Pr(\phi_p(\mathsf{s}, \mathsf{o}) = 1|\Theta) = \sigma(\eta_{spo}) = \sigma(\boldsymbol{r}_p^\top(\boldsymbol{e}_s \circ \boldsymbol{e}_o))$$

$\Pr(\Phi_p(s, o) = 1|\Theta)$ gives the probability of the triplet belonging to the knowledge graph. σ denotes the sigmoid function. $\sigma(x) = \frac{1}{1+\exp(-x)}$. $\mathbf{r_p}$ is the relational vector/matrix. $\mathbf{e_s}$ denotes the embedding of subject entity. $\mathbf{e_o}$ denotes the embedding of object entity. o is the most important operation in compositional based model, it is called compositional

operator. Different models use different compositional operator rest of the equation is pretty much the same. HolE(Nickel et al. 2016) uses circular correlation. RESCAL (Nickel et al. 2011) uses Tensor multiplication as a compositional operator.

3.3.1 RESCAL

RESCAL(Nickel et al. 2011) is based on tensor multiplication. A tensor is a multidimensional array. A tensor of 1-dimension is known as a vector, a tensor of 2-d is known as a matrix. Figure 3.17(Nickel et al. 2011) shows the tensor model of the order $(n \times n \times m)$, where n is the number of entity and m is the number of predicate/relation. let $\mathcal{X}$ denotes this tensor model. $\mathcal{X}_{ijk} = 1$ if (E_i, E_j, R_k) belongs to the knowledge graph, otherwise it is 0.

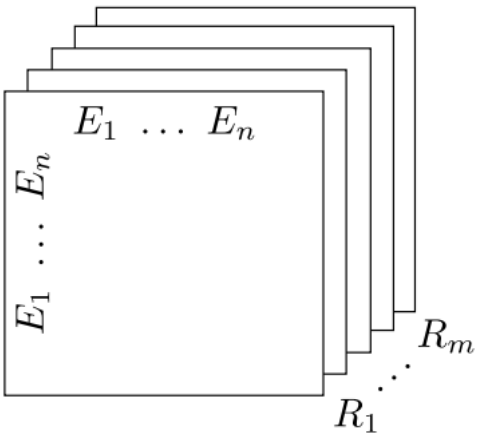


Figure 3.17: Tensor Model

$\mathcal{X}_k$ denotes the kth frontal slice, basically all the information of the kth relation (which pair of entity are related with the relation (R_k).

$$\mathcal{X}_k = AR_kA^T, \ for\ every\ k = 1,\ 2,\ 3, ..., m$$

We want every frontal slice $(\mathcal{X}_k)$ to be equal to AR_kA^T, where A is the latent representation of the entities, A is of order of $n \times m$. Every entity have the representation in A matrix. R_k is the relational matrix of order $m \times m$. Our goal is to make this interaction of entities and relation to be equal to the frontal slice $(\mathcal{X}_k)$.

Training

We need a loss/objective function for the purpose of the training(Nickel et al. 2011).

$$\min_{A,R_k} f(A,\ R_k)\ +\ g(A,\ R_k) \qquad (Objective\ Function)$$

$$f(A, R_k)\ =\ \frac{1}{2}\left(\sum_k \|\mathcal{X}_k\ -\ AR_kA^T\|_F^2\right) \qquad (Main\ Objective)$$

$$g(A, R_k)\ =\ \frac{1}{2}\lambda\left(\|A\|_F^2\| \ +\ \sum_k \|R_k\|_F^2\right) \qquad (Regularization\ to\ avoid\ overfitting)$$

$$f(A, R_k)\ =\ \frac{1}{2}\sum_{i,j,k}\left(\mathcal{X}_{ijk}\ -\ \mathbf{a}_i^T R_k \mathbf{a}_j\right)^2 \qquad (L2\ norm\ expanded)$$

a_i is the i_{th} row of the matrix A. The rest of the objective is pretty clear. Now we can use SGD to train over model.

Miscellaneous

The evaluation result is given in the figure 3.15 (Lin et al. 2015). We can see that RESCAL performed very poorly compared to other knowledge graph embedding algorithms. The reason is that there are lot of parameters and many of which are redundant. Its difficult to train this model.

Sharp readers might have noticed that the above objective or the goal does not relate with the equation described in section 3.3 and even sharper reader might have noticed that no use of tensor product is done in the above objective function. To answer both function first we have to understand the formula of tensor product between two vectors **a** *and* **b** (Kolda et al. 2009).

$$\mathbf{a}\ \otimes\ \mathbf{b}\ \in\ \mathbb{R}^{d^2} \qquad (The\ result\ is\ a\ matrix\ of\ order\ d \times d)$$

$$[\mathbf{a}\ \otimes\ \mathbf{b}]_{ij}\ =\ a_i b_j \qquad (Every\ component\ of\ b\ is\ multiplied\ by\ every\ component\ of\ a)$$

Now with some manuplication we can see that $\mathbf{r_p}^T(\mathbf{e}_s \otimes \mathbf{e}_o)\ =\ \mathbf{e}_s^T R_p \mathbf{e}_o$. The compositional operator is replaced with the tensor product. Rest of the details are same for all compositional based models. Which is pretty cool by only changing one operator we can create whole set of new knowledge embedding algorithms. The main problem/issue with RESCAL is that it uses too many parameters ot represent relation. Every relation is represented by a matrix of order $m \times m$. For Every relation entities (both subject and object) have different representation. Which is fine for small knowledge graphs, but for large knowledge graphs with thousands of relation RESCAL fails, in other words RESCAL is not scalable.

3.3.2 HoLE

HoLE(Holographic Embeddings of Knowledge Graph)(Nickel et al. 2016) is another compositional-model based embedding which we will discuss in this report.

$$\Pr(\phi_p(\mathsf{s}, \mathsf{o}) = 1|\Theta) = \sigma(\eta_{spo}) = \sigma(\boldsymbol{r}_p^\top(\boldsymbol{e}_s \circ \boldsymbol{e}_o))$$

This is the common equation for all the compositional based model. Different models differ in only the compositional operator (o). RESCAL uses tensor product as a compositional operator. The problem is that if we use tensor product as compositional operator then $\mathbf{r}_p$ should be of order $d^2 \times 1$ as $\mathbf{a} \otimes \mathbf{b} \in \mathbb{R}^{d^2}$. Number of parameter is of order of 2 (relation/predicate embedding).

We need to find the compositional operator that gives the output of the same dimension as the vectors. Most simple operations that can do is **concatenation, projection and non-linearity** (Nickel et al. 2016). We first concatenate two vectors, then project the vector to get desired dimension, and then apply non-linearity.

$$\mathbf{a} \circ \mathbf{b} = \psi(W(\mathbf{a} \oplus \mathbf{b}))$$
$$[\psi(W(\mathbf{a} \oplus \mathbf{b}))]_i = \psi\left(\sum_j w^a_{ij} a_j + \sum_j w^b_{ij} b_j\right)$$

Above equations does the same, first concatenate ($\oplus$) both the vectors, the project the vector (W) and then apply some non-linearity (ψ). We get vector of desired size so parameters required for $\mathbf{r}_p$ will also be less. There is a problem in this approach also, the interaction between the two vectors is very less compared to tensor product. This is also not a desired composition operator

We need best of both the operators, number of interaction should match with tensor product, the dimension of the output should be of the dimension of the input vectors. The compositional operator which HoLE uses is circular correlation operation.

$$\mathbf{a} \circ \mathbf{b} = \mathbf{a} \star \mathbf{b} \quad (where \star : \mathbb{R}^d \times \mathbb{R}^d \implies \mathbb{R}^d)$$
$$[\mathbf{a} \star \mathbf{b}]_k = \sum_{i=0}^{d-1} a_i b_{(k+i) mod\, d}$$
$$\mathbf{Pr}(\phi_p(\mathbf{s}, \mathbf{o})) = \sigma(\mathbf{r_p^T}(\mathbf{e_s} \star \mathbf{e_o}))$$

These are the equations for HoLE embeddings. The dimension of the output vector is same as the input vectors. The interaction between the elements of input vector is same as the tensor product.

Comparison between RESCAL and HoLE

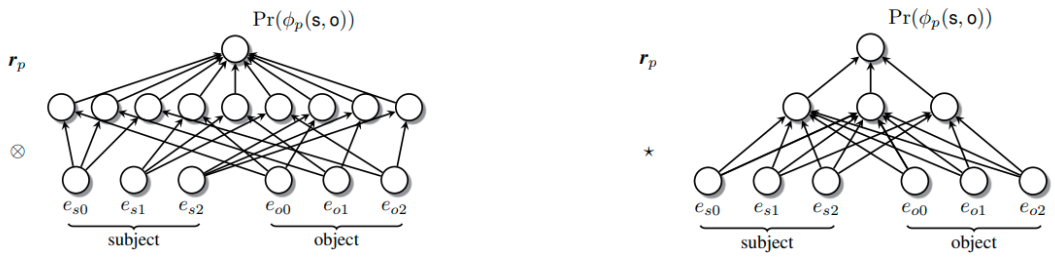


Figure 3.18: Comparison between RESCAL and HoLE

Figure 3.18 (Nickel et al. 2016) shows RESCAL and HoLE as neural networks for comparison purposes. Subject (head entity) and object (tail entity) are of 3 dimensions in

the above figure. In RESCAL the output is of 9 (3*3) dimension, then the hidden layer output is multiplied element wise with $\mathbf{r_p}$ to get the probability. Whereas in HoLE the output is of 3 dimensions (because of circular correlation operator), then the hidden layer output is multiplied element wise with $\mathbf{r_p}$ to get the probability.

Operator	$\circ$	**Memory** r_p	**Runtime** $r_p^\top (e_s \circ e_o)$
Tensor Product	$\otimes$	$\mathcal{O}(d^2)$	$\mathcal{O}(d^2)$
Circular Correlation	$\star$	$\mathcal{O}(d)$	$\mathcal{O}(d \log d)$

Figure 3.19: Comparison between RESCAL and HoLE in terms of complexities

Figure 3.19(Nickel et al. 2016) shows the comparison between RESCAL and HoLE in terms of complexities. the time complexity of HoLE is $O(dlogd)$ because we can use fast Fourier transorm(FFT) to compute the circular correlation.

Training

We need a loss function for training. Cross entropy loss is the obvious choice as this is model as the classification problem. We need gradients for weight update rule.

$$
\begin{aligned}
f_{spo} &= \sigma(\mathbf{r}_p^T \star (\mathbf{e}_s \star \mathbf{e}_o)) \\
\frac{\partial f_{spo}}{\partial \theta} &= \frac{\partial f_{spo}}{\partial \eta_{spo}} \frac{\partial \eta_{spo}}{\partial \theta} \\
\frac{\partial \eta_{spo}}{\partial \mathbf{r}_p} &= \mathbf{e}_s \star \mathbf{e}_o \\
\frac{\partial \eta_{spo}}{\partial \mathbf{e}_s} &= \mathbf{r}_p \star \mathbf{e}_o \\
\frac{\partial \eta_{spo}}{\partial \mathbf{e}_o} &= \mathbf{r}_p * \mathbf{e}_o
\end{aligned}
$$

Where $*$ is the circular convolution operator defined as :

$$
[\mathbf{a} * \mathbf{b}]_k = \sum_{i=0}^{d-1} a_i b_{(k-i) mod\, d}
$$

Above equations can be derived by using circular correlation and circular convolution properties for more information refer (Nickel et al. 2016).

Evaluation Results on Link Prediction Task

Link prediction task is described in section 3.2.1 . Datasets used for this experiment are same as the datasets used in the case of TransH (see section 3.8). Figure 3.2.1 shows the evaluation result on link prediction task.

	WN18					FB15k				
	MRR		Hits at			MRR		Hits at		
Method	Filter	Raw	1	3	10	Filter	Raw	1	3	10
TransE	0.495	0.351	11.3	88.8	94.3	0.463	0.222	29.7	57.8	**74.9**
TransR	0.605	0.427	33.5	87.6	94.0	0.346	0.198	21.8	40.4	58.2
ER-MLP	0.712	0.528	62.6	77.5	86.3	0.288	0.155	17.3	31.7	50.1
Rescal	0.890	0.603	84.2	90.4	92.8	0.354	0.189	23.5	40.9	58.7
HoLE	**0.938**	**0.616**	**93.0**	**94.5**	**94.9**	**0.524**	**0.232**	**40.2**	**61.3**	73.9

Figure 3.20: Results for Link Prediction Task

Metric used in MRR (Mean reciprocal rank) (higher the better) and Hits at (higher the better). HoLE perform best in WN18 dataset. In FB15K data set Hits@10 is best for TransE algorithm. MRR, Hits@1, Hits@3 are best for HoLE algorithm. Overall HoLE is consistent and best in terms of scalibility and performance.

3.4 Neural Network-Based Models

Almost every problem in NLP have neural network based solutions. Knowledge graph embeddings are no different. In this section we will discuss four neural based models. section 3.4.1 discusses the single layer model (which is linear layer with activation function)(Socher et. al. 2013). Second section 3.4.2 discusses the neural tensor network model (NTN) (Socher et. al. 2013), Third 3.4.3 and fourth 3.4.4 section discusses the convolution based neural network architecture.

3.4.1 Single Layer Model

The basic idea about every knowledge graph embedding algorithms are same. There is some score function lets say g which takes three vectors as input (h, r, t) and give some score depending on whether the triplet belongs to knowledge graph or not. Neural network models are no different, they all use some type of scoring function to train the knowledge graph embeddings.

As the name suggests single layer models contains a single layer. This is very simple neural network based model, it only contains a single layer followed by an activation function.

$$\begin{aligned} g(e_1, R, e_2) &= u_R^T f\left(W_{R,1} e_1 + W_{R,2} e_2\right) \\ &= u_R^T f\left(\left[W_{R,1}\, W_{R,2}\right] \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}\right) \end{aligned}$$

f is an activation function ($tanh$). $W_{R,1}$,$W_{R,2}$ $\in \mathbb{R}^{k \times d}$ and $u_R \in \mathbb{R}^{k \times 1}$ are the parameters of the relation R (Socher et. al. 2013). e_1 and e_2 are the embeddings for the entities and they are also the trainable parameters of the model. There is an interaction between the head entity and the tail entity but it is limited by the power of the activation function (in this case $tanh$).

3.4.2 Neural Tensor Network

Section 3.4.1 discusses the single layer model. The limitation of the single layer model is that the interaction between the head and tail entity is limited by the activation function. Neural tensor network solves this problem by cleverly designing the score function $g(e_1, R, e_2)$. First let us discuss the neural tensor model then we will discuss how it can solve this problem.

Figure 3.21 (Socher et. al. 2013) gives the overview of the model. For the triplet (*Bengal Tiger*, *has part*, *tail*). First the word embeddings of bengal and tiger are added to form the entity embedding e_1, similarly for the tail entity e_2. Both e_1 and e_2 are given to the neural tensor network (which is parameterized on the basis of relation R) as the input. The output is the score for the triplet i.e. whether the triplet belongs to the knowledge graph or not.

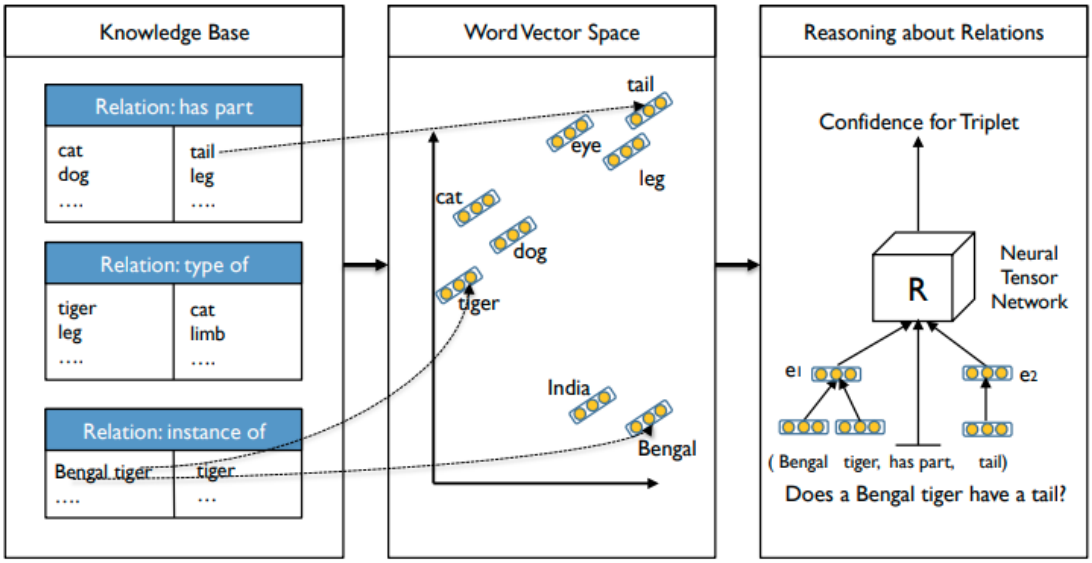


Figure 3.21: Overview of the model

Score Function

The most important part of any knowledge graph embedding is the score function. Score function is used in the loss which is used for training the network. The score function for the NTN model is defined as :

$$g(e1, R, e2) = u_R^T f\left(e_1^T W_R^{[1:k]} e_2 + V_R \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + b_R\right)$$

f is an activation function ($tanh$) which is applied element-wise. There are k slices (tensors), the parameters are represented by $W_R^[1:k]$. Bilinear tensor product is taken between the entities (head and tail) and the tensors. $W_R^[1:k] \in \mathbb{R}^{d \times d \times k}$ is a tensor. The bilinear tensor product $e_1^T W_R^{[1:k]} e_2$ results in vector of dimensions k ($h \in \mathbb{R}^k$). Each element (i) of the vector h is computed by $e_1^T W_R^{[i]} e_2$ bilinear tensor product. V_R and u_R are also the parameter of the model.

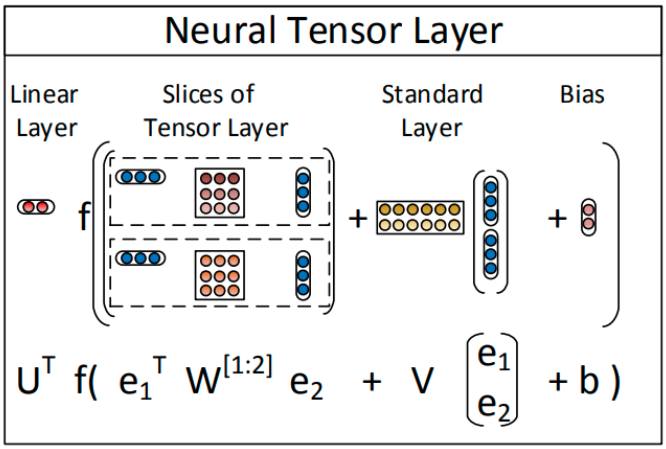


Figure 3.22: Visualization for the score function

Figure 3.22 (Socher et. al. 2013) gives an example of the score function in visualization form. The most important thing is the slices of the tensor. There are two slices in this example. Notice that in single layer model there was very limited interaction between the head and tail entities. NTN uses the bilinear tensor product ($e_1^T W_R^{[1:k]} e_2$) which increases the interaction between the entities.

Single layer model (slm) is the special case of the neural tensor network model if the tensors are set to 0.

Training

Training objective is similar to the above knowledge graph embeddings approaches. Margin based loss is used as a training objective.

$$J(\mathbf{\Omega}) = \sum_{i=1}^{N} \sum_{c=1}^{C} \max\left(0, 1 - g\left(T^{(i)}\right) + g\left(T_c^{(i)}\right)\right) + \lambda \|\mathbf{\Omega}\|_2^2$$

Goal is that the score function should give more score to the correct triplets and low score to corrupted triplet (for details on how to generate corrupted triplet see section 3.2.1).

$$\frac{\partial g(e_1, R, e_2)}{\partial W^{[j]}} = u_j f'(z_j) e_1 e_2^T, \quad \text{where } z_j = e_1^T W^{[j]} e_2 + V_j. \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + b_j,$$

Now using these equations we can easily train our network using backpropogation and gradient descent algorithm.

Datasets

Datasets used for the experiment is Wordnet and Freebase. Figure 3.23(Socher et. al. 2013) gives the statistics of the dataset.

Dataset	#R.	# Ent.	# Train	# Dev	# Test
Wordnet	11	38,696	112,581	2,609	10,544
Freebase	13	75,043	316,232	5,908	23,733

Figure 3.23: Dataset Statistics

We can see that both of these dataset contains very less number of relations. The reason is NTN is not scalable as some of the other knowledge graph embeddings algorithm. This makes this algorithm less useful for practical purposes.

Evaluation Result on Relation Triplets Classification Task

Triplet classification is a binary classification task. The task is that given a triplet (e_1, R, e_2) the task is to classify this triplet into two categories (whether this triplet belongs to knowledge graph or not).

NTN has a score function $g(e_1, R, e_2)$ and based on the threshold T_R (tuned on the development set), we decide whether the triplet belongs to knowledge graph or not. If $g(e_1, R, e_2) > T_R$ then the triplet belongs to knowledge graph otherwise not.

Model	WordNet	Freebase	Avg.
Distance Model	68.3	61.0	64.7
Hadamard Model	80.0	68.8	74.4
Single Layer Model	76.0	85.3	80.7
Bilinear Model	84.1	87.7	85.9
Neural Tensor Network	**86.2**	**90.0**	**88.1**

Figure 3.24: Accuracy on the Triplet Classification Task

Figure 3.24 (Socher et. al. 2013) gives the evaluation result. We can see that NTN performed better than the single layer model described above 3.4.1. The reason is the less interaction among the head and the tail entity.

3.4.3 Conv2d/ConvE

Conv2d/ConvE is based on the convolution neural network. The basic idea is still the same that there is a score function. The goal of the knowledge graph embedding algorithm is to increase the score on the correct triplet (h, r, t) and decrease the score on the corrupted triplets or vice versa.

There is also a secondary goal. The goal is to increase the interaction between the entities and the relation. This increase resulted in better performance in NTN (Neural Tensor Netwrk) (Socher et. al. 2013).

1D vs 2D Convoultion

This knowledge graph embedding algorithm 2 dimensions convolutions are used to increase the interaction, to make this point clear let us take an example (Dettmers et. al. 2018).

let us assume that the entity and the relation are of four dimensions.

$$
\begin{aligned}
h &= [a\ a\ a\ a] \\
t &= [b\ b\ b\ b] \\
[h:t] &= [a\ a\ a\ a\ b\ b\ b\ b] \quad \textit{Concatenation}
\end{aligned}
$$

Now if we assume the kernel of size k, then there will be $O(k)$ interaction among these entities and the relations. If we take a matrix with same 4 elements the number of interaction in the $O(m * n * k)$. Where m and n are the dimension of the matrix.

$$
\left(\begin{bmatrix} a & a \\ a & a \end{bmatrix} ; \begin{bmatrix} b & b \\ b & b \end{bmatrix} = \begin{bmatrix} a & a \\ b & b \\ a & a \\ b & b \end{bmatrix} \right)
$$

In the above case if we assume the kernel size to be k. The order of matrix as $m \times n$ then the number of interaction between these two entities is at order of $m * n * k$. This concept is used in Conv2d to increase the interaction.

Score Function

The score function in Conv2d case is defined by the neural network. Figure 3.25 (Dettmers et. al. 2018) gives the architecture for the computaion of the score function.

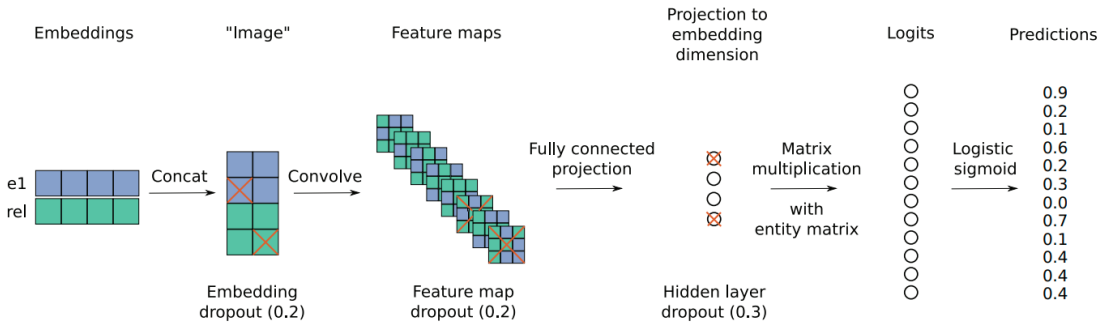


Figure 3.25: Architecture for the score function

First the entity and relation embeddings are concatenated in the method discussed above. The convolution layer is applied to the two dimensional concatenation. The feed forward layer is applied to the feature maps. The dot product of this representation is taken with all the tail entity. The sigmoid layer is applied to the result of the dot product which gives the score for the triplet.

$$
\psi_r(\mathbf{e}_s, \mathbf{e}_o) = \quad f(\mathrm{vec}(f([\overline{\mathbf{e}_s}; \overline{\mathbf{r}_r}] * \omega))\mathbf{W})\mathbf{e}_o
$$

$\psi_r(e_s, e_o)$ is the score function, $\star$ is the convolution operator. This is the mathematical equation of the above architecture. The number of parameters are also linear in the

number of entity and the number of relation. Which means that it is a scalable algorithm which can be applied to large knowledge graphs.

Loss Function

This is modeled as a binary classification task. Binary cross entropy is used as the loss function for training the network.

$$L(p,t) = -\frac{1}{N}\sum_{i}\left(t_i \log(p_i) + (1 - t_i)\log\left(1 - p_i\right)\right)$$

Backpropogation with the gradient descent is used to minimize this loss.

Inverse Model

Before discussing the evaluation results we should first discuss another model inverse model (Dettmers et. al. 2018). This is a rule based model which outperforms all other embedding algorithms in WN18 and FB15k datasets (see section 3.2.1). There are inverse relations present in the knowledge graphs for example *hypernym* and *hyponym* (Dettmers et. al. 2018). If $(crow, hyponym, bird)$ is the triplet present in the knowledge graph then $(bird, hypernym, crow)$ is also the correct triplet (as *hyponym* and *hypernym* are the inverse relation).

The authors (Dettmers et. al. 2018) observed that there are such inverse relations in the test set (in WN18 and FB15k Datasets) and any model can exploit this inverse relation property and get good result. To get the sense of how big this problem is authors reported the result of link prediction task.

	WN18					**FB15k**				
			Hits					Hits		
	MR	MRR	@10	@3	@1	MR	MRR	@10	@3	@1
DistMult (Yang et al. 2015)	902	.822	.936	.914	.728	97	.654	.824	.733	.546
ComplEx (Trouillon et al. 2016)	–	.941	.947	.936	.936	–	.692	.840	.759	.599
Gaifman (Niepert 2016)	**352**	–	.939	–	.761	75	–	.842	–	.692
ANALOGY (Liu, Wu, and Yang 2017)	–	**.942**	.947	.944	**.939**	–	.725	.854	.785	.646
R-GCN (Schlichtkrull et al. 2017)	–	.814	.964	.929	.697	–	.696	.842	.760	.601
ConvE	504	**.942**	.955	.947	.935	**64**	**.745**	**.873**	**.801**	.670
Inverse Model	567	.861	**.969**	**.968**	.764	1897	.706	.737	.718	**.689**

Figure 3.26: Link Prediction result on WN18 and FB15k Datasets

Figure 3.26 (Dettmers et. al. 2018) shows the result of link prediction task on WN18 and FB15k datasets. We can see that the normal rule based inverse model beats all other knowledge graph embedding algorithm. This shows that there is a big problem with these datasets. Authors (Dettmers et. al. 2018) proposed WN18RR which is a dataset in which the problem of inverse model is taken care of.

Evaluation Result on Link Prediction Task

The datasets used for evaluation are WN18RR and FB15k-237. Which are similar to WN18 and FB15k with inverse relations removed from the test set.

	WN18RR					FB15k-237				
			Hits					Hits		
	MR	MRR	@10	@3	@1	MR	MRR	@10	@3	@1
DistMult (Yang et al. 2015)	**5110**	.43	.49	.44	.39	254	.241	.419	.263	.155
ComplEx (Trouillon et al. 2016)	5261	.44	**.51**	**.46**	**.41**	339	.247	.428	.275	.158
R-GCN (Schlichtkrull et al. 2017)	–	–	–	–	–	–	.248	.417	.258	.153
ConvE	5277	**.46**	.48	.43	.39	**246**	**.316**	**.491**	**.350**	**.239**
Inverse Model	13219	.36	.36	.36	.36	7148	.009	.012	.010	.006

Figure 3.27: Link Prediction result on WN18RR and FB15k-237 Datasets

Figure 3.27 (Dettmers et. al. 2018) show the result. We can see that inverse model performed poorly on these new datasets. The metrics used are MR (see section 3.2.1), MRR and Hits@k (see section 3.9). ConvE perfoms good at almost all the metric and database. The reason is the more interaction between the entities and the relations.

3.4.4 ConvKB

ConvKB (Nguyen et. al. 2017) is also based on the convolution neural network. The basic idea still remains the same. The goal is to compute a score function which takes a triplet (h, r, t) as the input and gives the score as the output. ConvKB is very similar to ConvE and yet it gives better results then ConvE.

ConvE apply the convolution operation to only the head entity embeddings and the relation/predicate embedding. ConvKB applies the same convolution operator to head entity, relation entity and the tail entity. This is the only difference between ConvKB and ConvE rest of the details are pretty much the same.

Score Function

The score function in ConvKB case is defined by the neural network. Figure 3.28 (Nguyen et. al. 2017) shows the architecture of the score function.

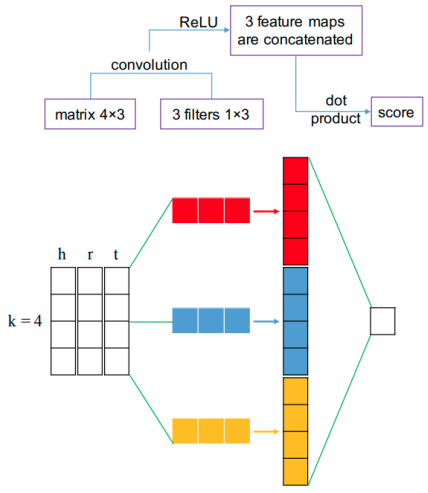


Figure 3.28: Score Function

In this example the embedding size is assumed to be four. First head entity, relation/predicate, tail entity are concatenated vertically thus forming matrix of size $e \times 3$. Convolution is applied with three kernel size of 1×3 resulting in three vectors of dimension four (size of the embedding). The result (feature maps) are concatenated. The dot product between the w vector and the concatenated vector to give the score.

$$f(h,r,t) \;=\; concat\left(g\left(\left[\mathbf{v}_h, \mathbf{v}_r, \mathbf{v}_t\right] \star \mathbf{\Omega}\right)\right).\mathbf{w}$$

($f(h,r,t)$ is the score function. This is the mathematical equation for the above architecture.

Dataset

The dataset used for the experiment are WN18RR and FB15k-237. The statistics of the datasets are given the the figure 3.29.

Dataset	$\mid \mathcal{E} \mid$	$\mid \mathcal{R} \mid$	#Triples in train/valid/test		
WN18RR	40,943	11	86,835	3,034	3,134
FB15k-237	14,541	237	272,115	17,535	20,466

Figure 3.29: Dataset Statistics

WN18RR contains 11 relations (same as WN11) and 40,943 enitities. FB15k-237 contains 237 relations which are relatively small compared to FB15k dataset. FB15k-237 doesnot have the inverse relation problem while FB15k have, so WN18RR and FB15k-237 are the right dataset to evaluate the knowledge graph embedding algorithm.

Loss Function

The goal of the objective function/loss function is similar to all the knowledge graph embedding algorithm. The goal is to increase the score for the correct triplet and decrease the score for the corrupted triplet or vice versa.

$$\mathcal{L} = \sum_{(h,r,t)\in\{\mathcal{G}\cup\mathcal{G}'\}} \log\left(1+\exp\left(l_{(h,r,t)}\cdot f\left(h,r,t\right)\right)\right) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

$$\text{in which, } l_{(h,r,t)} = \begin{cases} 1 \text{ for } (h,r,t) \in \mathcal{G} \\ -1 \text{ for } (h,r,t) \in \mathcal{G}' \end{cases}$$

G is the set of the correct triplet i.e. the triplet present in the knowledge graph and G‘ is the set of the corrupted triplet i.e. the triplet that does not belong to the knowledge graph. The goal is to decrease the score for the correct triplet and increase the score fir the corrupted triplet.

Evaluation Result on Link Prediction Task

The results of the experiment are shown in the figure 3.30 (Nguyen et. al. 2017) . Metric used are MR (see section 3.2.1), MRR, Hits@10 (see section 3.2.2).

Method	WN18RR			FB15k-237		
	MR	MRR	H@10	MR	MRR	H@10
IRN (Shen et al., 2017)	–	–	–	211	–	46.4
KBGAN (Cai and Wang, 2018)	–	0.213	48.1	–	0.278	45.8
DISTMULT (Yang et al., 2015) [⋆]	5110	0.43	49	254	0.241	41.9
ComplEx (Trouillon et al., 2016) [⋆]	5261	0.44	51	339	0.247	42.8
ConvE (Dettmers et al., 2018)	5277	**0.46**	48	246	0.316	49.1
TransE (Bordes et al., 2013) (our results)	3384	0.226	50.1	347	0.294	46.5
Our ConvKB model	**2554**	0.248	**52.5**	257	**0.396**	**51.7**
KB_{LRN} (García-Durán and Niepert, 2017)	–	–	–	**209**	0.309	49.3
R-GCN+ (Schlichtkrull et al., 2017)	–	–	–	–	0.249	41.7
Neural LP (Yang et al., 2017)	–	–	–	–	0.240	36.2
Node+LinkFeat (Toutanova and Chen, 2015)	–	–	–	–	0.293	46.2

Figure 3.30: Evaluation Result on the Link Prediction Task

The important thing to notice is that ConvKB performs better than ConvE in almost every metric and dataset. The difference between these two models is very less then also ConvKB performs better than ConvE. According to me the reason for this result is that, only head entity and the relation interacted in the case of ConvE, while head entity, relation entity and the tail entity interacted in the case of ConvKB.

3.5 Open KG Embedding

Open Knowledge Graph (OpenKG) consists of (*noun phrase*, *relation phrase*, *noun phrase*). OpenKG are constructed using open IE extraction methods. As these are automatically constructed so they are often not canonocalized i.e. some entities are represented by different names e.g. Narendra Modi and Modi, both refers to the same person but they will be stored twice. Which can lead to storage problem and will return the incomplete facts. for example Narendra Modi is the PM of India is one of the fact in open kg. Modi is the son of Heerben Modi is also one of the fact in the open kg. If we ask information about Modi with this knowledge graph it will give incomplete information. As for this knowledge graph Narendra Modi and Modi are the different entity. Merging these entities are extremely important for the knowledge graph section 3.5.1 discusses one technique to canonicalize OpenKg. Section 3.5.2 discusses an embedding algorithm to train the knowledge graph embedding for OpenKG. Section 3.5.2 uses the concepts discussed in section 3.5.1.

3.5.1 CESI: Canonicalizing Open Knowledge Bases using Embeddings and Side Information

CESI (Vashishth et. al. 2019) uses both the knowledge graph embeddings and some side information to find out which entities/relations are similar to one another.

Canonicalizing Open Knowledge Bases is very important task it not only saves the storage space, also it helps other algorithms to return the complete facts.

Architecture

The approach is divided into three main steps :

- **Side Information :** The first step is to collect the side information like entity linking, IDF token overlap, PPDB information, WordNet information, Morp Normalization. These are the side information for the entity. The side information about the relation are PPDB information, WordNet side information, AMIE information and KBP information. For more information about these side information please see Vashishth et. al. 2019.
- **Embedding NP and Relation Phrases :** The embedding of noun phrase and relation phrase are learned using any one of the algorithm discussed above (authors (Vashishth et. al. 2019) used HoLE (section 3.3.2) algorithm) along with the side information gathered by the above step.
- **Clustering Embeddings and Canonicalization :** This is the last step of this algorithm. The embeddings gathered from the above algorithm are clustered together with the help of Hierarchical Agglomerative Clustering (HAC) algorithm. The entity closer to the mean of the cluster is chosen as the name for the cluster i.e. all the entity inside the cluster will have the this chosen name.

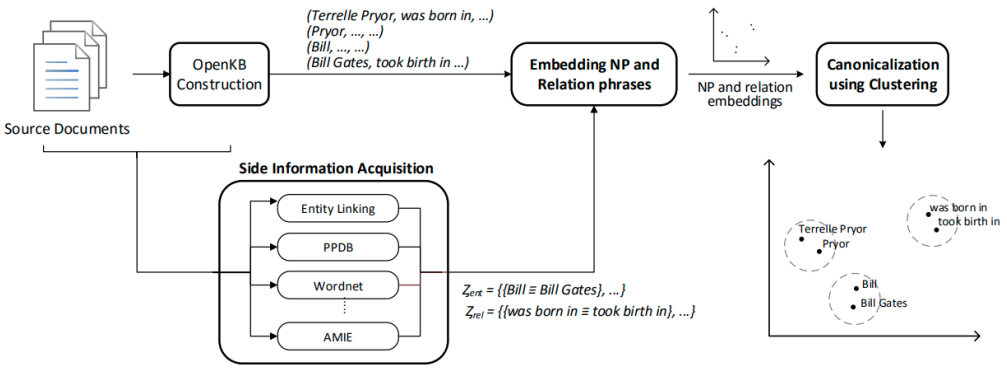


Figure 3.31: Architecture of the CESI

Figure 3.31 (Vashishth et. al. 2019) show the architecture of CESI algorithm. First side information is gathered using the source document. Embeddings are learned using the side information and the HoLE (section 3.3.2) algorithm. The third step is to use the clustering algorithm to make clusters of similar embeddings. The second step of this algorithm (learning the embedding) is the most important one. It is described in the next section.

Embedding Noun Phrases and Relation Phrases

Knowledge graph embedding algorithm used by the authors is HoLE algorithm along with using the side information. Side information is combined with the loss function of the

HoLE algorithm.

Embeddings are calculated by only using side information, then these embeddings are applied as a soft constraints in the loss function. First part of the loss function is same as the HoLE algorithm (for more information see section 3.3.2). Second part is the soft constraints applied to this loss function (using the side information). Third part is the regularizer to avoid the overfitting.

$$\begin{aligned}\min_{\Theta}\ & \lambda_{str} \sum_{i \in D_+} \sum_{j \in D_-} \max(0, \gamma + \sigma(\eta_j) - \sigma(\eta_i)) \\ & + \sum_{\theta \in \mathscr{C}_{\text{ent}}} \frac{\lambda_{\text{ent},\theta}}{|\mathcal{Z}_{\text{ent},\theta}|} \sum_{v, v' \in \mathcal{Z}_{\text{ent},\theta}} \|e_v - e_{v'}\|^2 \\ & + \sum_{\phi \in \mathscr{C}_{\text{rel}}} \frac{\lambda_{\text{rel},\phi}}{|\mathcal{Z}_{\text{rel},\phi}|} \sum_{u, u' \in \mathcal{Z}_{\text{rel},\phi}} \|r_u - r_{u'}\|^2 \\ & + \lambda_{\text{reg}} \left(\sum_{v \in V} \|e_v\|^2 + \sum_{r \in R} \|e_r\|^2 \right).\end{aligned}$$

We can use the backpropogation with gradient descent to minimize the above loss function.

Datasets

Three datasets are used for the experiment purpose (Base, Ambigious, Reverb45K). Figure 3.32 (Vashishth et. al. 2019) shows the statistics of the dataset.

Datasets	# Gold Entities	#NPs	#Relations	#Triples
Base	150	290	3K	9K
Ambiguous	446	717	11K	37K
ReVerb45K	7.5K	15.5K	22K	45K

Figure 3.32: Dataset Statistics

Reverb45K is the largest dataset among three. It contains 7.5K gold entities (which is almost 16 times bigger than the ambigious dataset), 15.5K Noun Phrases, 22K Relation and 45K triplets.

Evaluation Metric

The evaluation metric used for reporting the experiment results are (let E denote the pure cluster and C denote the cluster predicted by the algorithm) :

- **Macro Precision and Recall :** Macro precision is defined as the ratio between the number of pure cluster (i.e. the clusters which do not contain element from any other gold cluster).

$$P_{\text{macro}}(C,E) = \frac{|\{c \in C : \exists e \in E : e \supseteq c\}|}{|C|}$$
$$R_{\text{macro}}(C,E) = P_{\text{macro}}(E,C)$$

- **Micro Precision and Recall :** Micro precision is defined as :

$$P_{\text{pair}}(C,E) = \frac{\sum_{c \in C} |\{(v,v') \in e, \exists e \in E, \forall (v,v') \in c\}|}{\sum_{c \in C} |c|C_2}$$
$$R_{\text{pair}}(C,E) = \frac{\sum_{c \in C} |\{(v,v') \in e, \exists e \in E, \forall (v,v') \in c\}|}{\sum_{e \in E} |e|C_2}$$

- **Pairwise Precision and Recall :** As the name suggests it calculated the pairwise precision and the recall. It is defined as :

$$P_{\text{pair}}(C,E) = \frac{\sum_{c \in C} |\{(v,v') \in e, \exists e \in E, \forall (v,v') \in c\}|}{\sum_{c \in C} |c|C_2}$$
$$R_{\text{pair}}(C,E) = \frac{\sum_{c \in C} |\{(v,v') \in e, \exists e \in E, \forall (v,v') \in c\}|}{\sum_{e \in E} |e|C_2}$$

Evaluation Result

The evaluation result on the three datasets discussed above is given in the figure 3.33 (Vashishth et. al. 2019).

Method	Base Dataset			Ambiguous Dataset			ReVerb45K			
	Macro	Micro	Pair.	Macro	Micro	Pair.	Macro	Micro	Pair.	Row Average
Morph Norm	58.3	88.3	83.5	49.1	57.2	70.9	1.4	77.7	75.1	62.3
PPDB	42.4	46.9	32.2	37.3	60.2	69.3	46.0	45.4	64.2	49.3
EntLinker	54.9	65.1	75.2	49.7	83.2	68.8	62.8	81.8	80.4	69.1
Galárraga-StrSim	88.2	96.5	97.7	66.6	85.3	82.2	69.9	51.7	0.5	70.9
Galárraga-IDF	94.8	97.9	98.3	67.9	82.9	79.3	71.6	50.8	0.5	71.5
Galárraga-Attr	76.1	51.4	18.1	**82.9**	27.7	8.4	**75.1**	20.1	0.2	40.0
GloVe	95.7	97.2	91.1	65.9	89.9	90.1	56.5	82.9	75.3	82.7
HolE (Random)	69.5	91.3	86.6	53.3	85.0	75.1	5.4	74.6	50.9	65.7
HolE (GloVe)	75.2	93.6	89.3	53.9	85.4	76.7	33.5	75.8	51.0	70.4
CESI	**98.2**	**99.8**	**99.9**	66.2	**92.4**	**91.9**	62.7	**84.4**	**81.9**	**86.3**

Figure 3.33: Evaluation Result

We can see that CESI performed better than all the other algorithms. The reason is that it uses side information to learn the embedding of the knowledge graph.

3.5.2 CaRe: Open Knowledge Graph Embeddings

All the embedding algorithms which we discussed above were focused on the embeddings of the ontology. We can use that algorithms for OpenKG also but they may not give good

results. CaRe is an knowledge graph embedding algorithm which is designed for OpenKG and it also give good results compared to other embedding algorithms.

There is a problem with OpenKG. OpenKGs are constructed using the Open Information Extraction (OpenIE) algorithms. There are entities which have same semantic meaning but they are represented by different nodes. This problem is depicted in figure **??** (Gupta et. al. 2019). Barack and Barack Obama refer to the same entity and yet they are represented by different node. This leads to redundant storage. Some information is also lost.

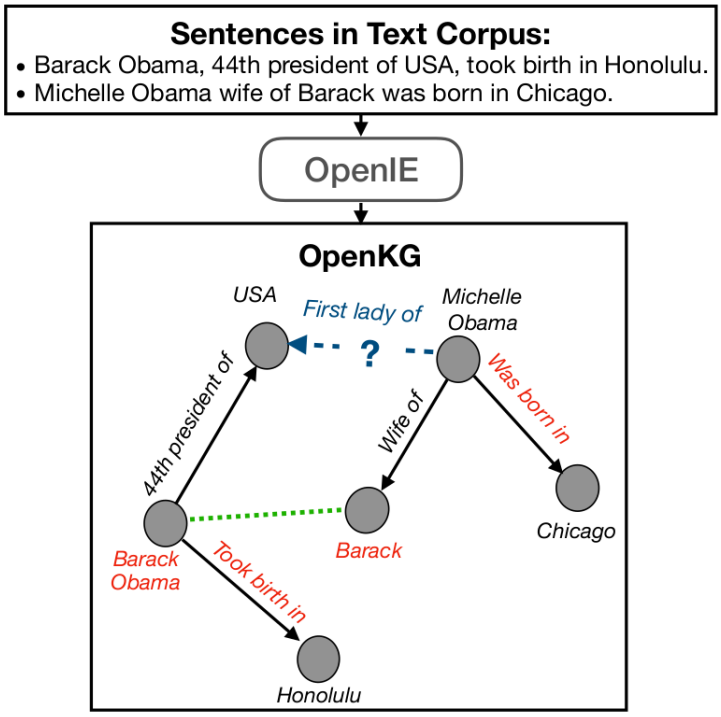


Figure 3.34: OpenKG Problem

To solve this problem we have a simple option, use the CESI algorithm described above 3.5.1, merge the nodes which belong to same cluster. There is a problem in this approach, CESI is not a perfect algorithm. There is an error assosiated with the clustering. This error will be propogated to the downstream tasks such as QA. How CaRe solves this problem we will see in the next section.

CaRe Steps

CaRe (Gupta et. al. 2019) algorithm is divided into two steps :

- **Canonicalization Augmentation:** CESI algorithm is used to get the clusters of the noun phrase. Nodes are not merged because of the reason discussed above. Instead undirected edges are added between the nodes which belongs to same cluster. Figure 3.35 (Gupta et. al. 2019) shows an example of the first step. dotted line denotes the undirected edges, since USA, United States and United States of America belongs to the same cluster.

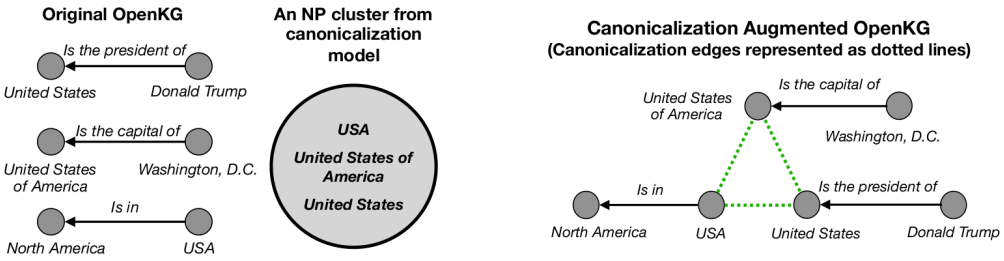


Figure 3.35: Step_1 of CaRe

- **Step 2** Step 2 is further divided into more steps. First step is to choose the base model (base knowledge graph embedding model), CaRe uses some other knowledge graph embedding algorithm. Base model can be any algorithm discussed above. Other steps are to design the Phrase Encoder Network (PN) and Canonical Cluster Encoder Network (CN) (which are described below). Figure 3.36 (Gupta et. al. 2019) shows the step 2 of the CaRe algorithm.

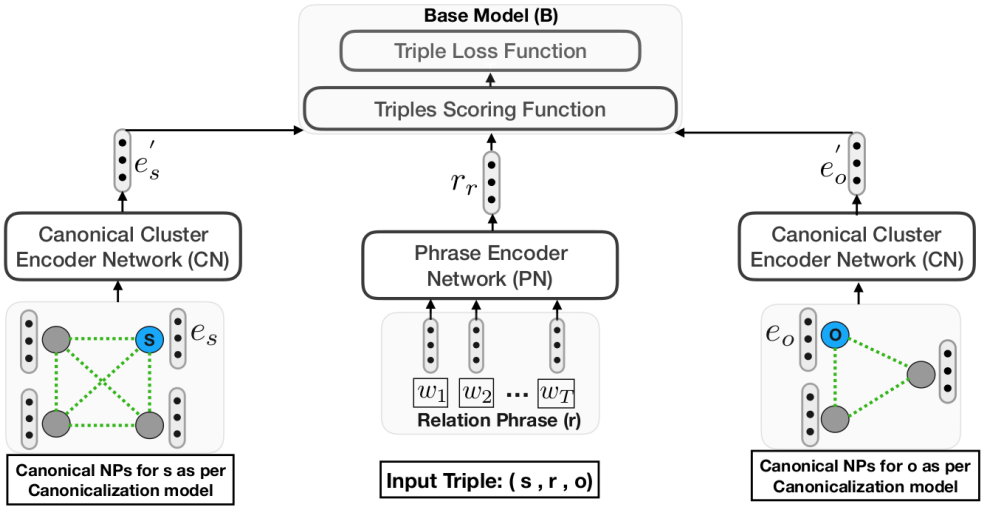


Figure 3.36: Step_2 of CaRe

  - **Phrase Encoder Network:** This is the BiGru network in which the relation phrase is tokenize into words. These tokens are given as input to the BiGru network. Last states of BIiGRU are concatenated to get r_r.
  - **Canonical Cluster Encoder Network:** First for each noun phrase (NP), the context vector is created e_n^c, which is the average over all the neighbours of the noun phrase connected by the undirected edge (dotted line).

$$e_n^c = \left(\sum_{i \in N(n) \frac{1}{|N(n)|} e_i} , \forall n \in N \right)$$

. $N(n) = \{i | i \in N, (n,i) \in C\}$ is the neighbour of the node n.

Now average is taken with this context vector and the embedding vector of the

node n.

$$e_n^{'} = \frac{e_n + e_n^c}{2}$$

The output of these vectors computed by phrase encoder network and canonical cluster encoder network are given to the base model and then the whole network is trained using the back propogation and the gradient descent.

Datasets

The datasets used in the experiment are shown in the figure 3.37 (Gupta et. al. 2019). They are similar to the datasets used for the CESI experiments.

Datasets	ReVerb45K	ReVerb20K
# NPs	27K	11.1K
# RPs	21.6K	11.1K
# Gold NP Clusters	18.6K	10.8K
# Train Triples	36K	15.5K
# Test Triples	5.4K	2.4K
# Validation Triples	3.6K	1.6K

Figure 3.37: Statistics Dataset

ReVerb20K is the subset of the ReVerb45K dataset with 11.1K noun phrase and the relation phrase.

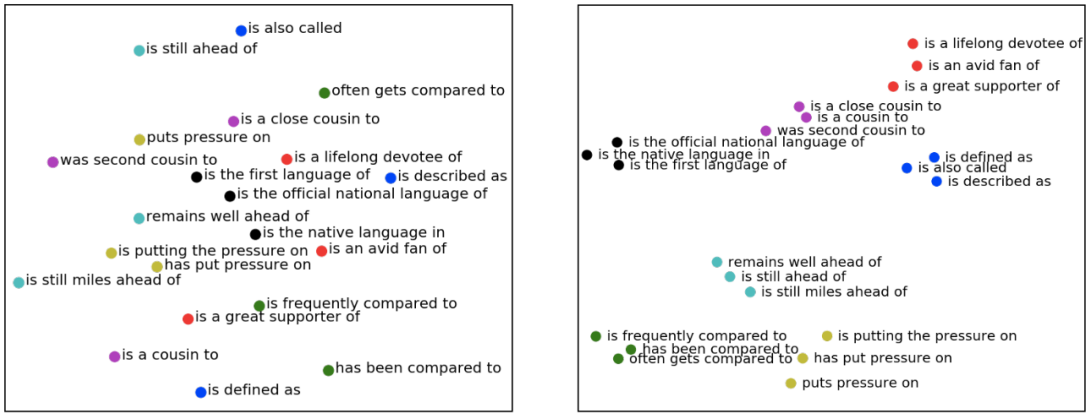
Evaluation Results on The Link Prediction Task

Link prediction task is described in section 3.2.1. Link prediction is performed by choosing base model as ConvE (see section 3.4.3). Base model can be any model discussed above but for experiment purposes authors (Gupta et. al. 2019) only explored base model as ConvE.

Method	ReVerb45K					ReVerb20K				
	MR	MRR	Hits@10	Hits@30	Hits@50	MR	MRR	Hits@10	Hits@30	Hits@50
TransE	2955.8	.193	.361	.446	.478	1425.8	.126	.299	.411	.468
TransH	2998.2	.194	.362	.442	.478	1464.4	.129	.303	.409	.467
DistMult	8988.8	.051	.051	.052	.065	6260.0	.033	.044	.055	.060
ComlEx	7786.5	.047	.047	.048	.073	5502.2	.037	.058	.075	.085
R-GCN	2866.8	.042	.046	.091	.113	1204.3	.122	.187	.263	.305
ConvE	2650.8	.233	.338	.401	.429	1014.5	.294	.402	.491	.541
CaRe(B=ConvE)	**1308.0**	**.324**	**.456**	**.543**	**.579**	**973.2**	**.318**	**.439**	**.525**	**.566**

Figure 3.38: Evaluation Result on Link Prediction Task

Figure 3.38 (Gupta et. al. 2019) shows the evaluation result on two datasets discussed above. We can easily see that CaRe(B=ConvE) (Base is ConvE) gives the best result among all the other models consistently.



(a) ConvE

(b) CaRe(B=ConvE)

Figure 3.39: Visualization (Relation Phrase)

Figure 3.39 (Gupta et. al. 2019) shows the visualization of the relation phrase embeddings generated by CaRe(B=ConvE) and the ConvE model. We can easily see that the similar phrases are closer to each other in the case of CaRe(B=ConvE) which shows that CaRe is also learning the similarity between the phrases.

3.6 Summary

In this chapter, we first discussed the intuition behind the knowledge graph embedding algorithms. The intuition is that the nodes having similar neighbours should have similar knowledge graph embeddings. We also discussed the importance of knowledge graph embeddings in the applications like link prediction, question answering. Next we disscused the translation based models, which uses vector addition (translation) between the head entity embeddings (or projection of head entity) and the relation embeddings to get the embeddings of the tail entity. In the translation based models we first discussed the TransE model which uses simple vector addition (translation) between the head entity embedding and the relation embedding to get the tail entity embedding ($\mathbf{h} + \mathbf{r} = \mathbf{t}$). Next we discussed TransH model, which projects the head and tail entity into the hyper-plane specific to the relations, then the translation is performed similar to the TransE model. TransR is very similar to TransH, instead of projecting the embeddings to the hyper-plane, TransR proposes that the entities and the relations should have different spaces. First both the head and tail entity are projected into different space specific to the relation and then the translation is performed similar to the TransE model.

In the second part of the chapter we discussed the compositional based models, which uses the compositional operator between the head enitity and the tail entity. we discussed RESCAL which uses tensor product as a compositional operator between the head and the tail entity. HoLE uses the circular correlation operator between the head and the tail entity. RESCAL algorithm have more trainable parameters (because of the tensor product), which makes the algorithm not scalable. HoLE have comparatively less number of parameter (comparable to TransE), so this makes HoLE algorithm more scalable. We also discussed different compositional operator that can be used. We discussed the concatenation, projection and non-linearity as the compositional operator, this have limitation of less interaction among the embeddings compared to the tensor product.

In the third part of the chapter, we discussed the neural network based models. We first discussed SLM model, which have very less number of interaction between the embeddings which resulted in the poor performance of the algorithm. We also discussed NTN model, which increases the interaction between the embeddings but increases the number of trainable parameter. This results that the algorithm is less scalable. Next we discussed the convolution network based knowledge graph embeddings. First we discussed Conv2d algorithm, which concatenates the entities embeddings (head entity and tail entity) in 2 dimension (to increase the interaction), then normal convolution architecture was used. ConvKB uses the similar idea but gets very good result, instead of concatenating only the entities embeddings. ConvKB concatenates all the three embeddings(head entity, relation,tail entity).

In the last part of the section, discussion was focused on the OpenKG. First we discussed the algorithm for canonicalizing OpenKG. Which uses the information from the source documents and the embedding of the knowledge graph. We then discussed the algorithm for knowledge graph embedding specific to the openKG. There are some challenges (phrases with same semantic meaning have different nodes in the graph) in OpenKG, that is why we needed different algorithm for openKG. CaRe uses the CESI algorithm along with any knowledge graph embedding algorithm. The main components of the CaRe algorithms are phrase encoder network (for encoding relation phrases) and canonical cluster encoder network (for encoding noun phrases).

Chapter 4

Application of Knowledge Graph Embedding : Question Answering

This chapter contains one of the most important application of the knowledge graph embeddings : Question Answering. Introduction about the problem and the small introduction of the Question Answering is given in the section 4.1. Section 4.2 discusses an algorithm that uses knowledge graph embeddings for question answering task. Subsection 4.2.1 contains the information about the datasets used for the task. Architecture of the model is explained in the subsection 4.2.2. The most important component of the architecture are Predicated and Head Entity learning model, and Head Entity Detection model, which are discussed in section 4.3 and the subsection 4.3.1 respectively. Joint distance metric which determines which fact should be chosen as the answer is described in section 4.3.2. Next we discuss the evaluation of the Question Answering Systems (Section 4.4). We discuss a transformer based approach for evaluating question answering systems in the section 4.4.1. The chapter is summarized in section 4.5.

4.1 Introduction

Question Answering is one of the important downstream task. There are many algorithms to solve this task (both statistical and deep learning). Usually a context/paragraph and question is given as an input to these models, then they give answers as an output. In this chapter we will discuss algorithm in which we give knowledge graph and a simple question an input, then get answers as an output. Section 4.2 discusses an algorithm that uses knowledge graph embeddings for question answering task.

Evaluation of question answering task is also very difficult task. There can be multiple answers correct, section 4.4 discusses the transformers based approaches to do evaluation of the question answering systems.

4.2 Knowledge Graph Embedding Based Question Answering

Knowledge Graph Embedding Based Question Answering (KGEQA) (Huang et. al. 2019) aims to solve the question answering task on the knowledge graph. The approach that

we will discuss here aims to answer simple questions on the knowledge graph. First we should understand what is the definition of the simple question.

The question which can be answer unambiguously if we know the head enitity and predicate. The answer is the tail entity. If we ask "When was Dhoni born?" from the knowledge graph. This can be answered if we know the head entity ("Dhoni") and the predicate ("born") then the answer is the tail entity ("7 July 1981"). This is the example of the simple question. We will discuss an algorithm (Huang et. al. 2019) to answer the simple question which uses the knowledge graph embeddings.

4.2.1 Datasets

Before understanding the algorithm, first we should understand the data which we need for training this algorithm. We need a knowledge graph (from which the questions will be answered), we need the simple questions (input to the model for training), we also need the triplet (head entity, predicate/relation, tail entity) for training the model.

	FB2M	FB5M	SimpleQuestions
# Training	14,174,246	17,872,174	75,910
# Validation	N.A.	N.A.	10,845
# Test	N.A.	N.A.	21,687
# Predicates (M)	6,701	7,523	1,837
# Entities (N)	1,963,130	3,988,105	131,681
Vocabulary Size	733,278	1,213,205	61,336

Figure 4.1: Statistics of the Dataset

Figure 4.1 (Huang et. al. 2019) show the statistics of the dataset. Freebase is used for the knowledge graph, the simple question which can be answered if know the head entity and the predicate of the free base is also given with the dataset.

Our goal is to find the head entity and the predicate from the knowledge graph, then we can easily find the tail entity and thus our answer.

4.2.2 Architecture

The basic idea of the algorithm is simple. We need head entity and predicate from the question. First knowledge graph is embedded into a vector of lower dimension. From the question we predict the head entity embedding and the predicate embedding. We can calculate/predict the tail entity embedding using the entity embedding and the predicate embedding (link prediction task). Now the triplet/fact (h, r, t) closest to this predicted triplet is chosen as the answer. Figure 4.2 (Huang et. al. 2019) shows the architecture of the model. We can easily see the above steps in pictorial format. Predicate learning, Head entity learning, Head entity detection model, finding the closest facts are the most important component of the model. In the next sections we will look into that.

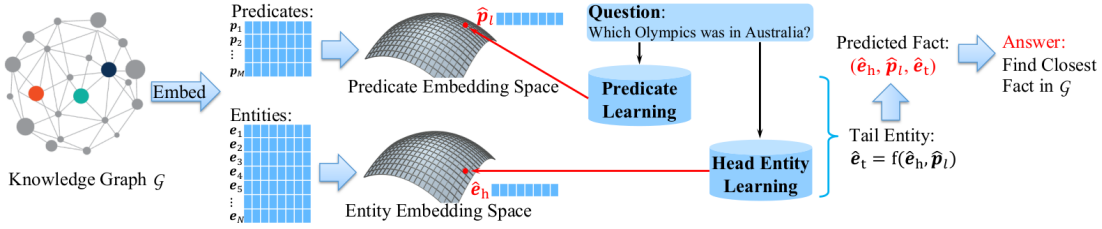


Figure 4.2: Overall Architecture

4.3 Predicate and Head Entity Learning Model

Basically what we have to do is give question as the input to the model. The supervision of the model is the predicate embedding (in the case of Predicate learning model) and Head entity (in the case of Head entity learning model).

We can use any neural network model, authors used Bi-directional LSTM with attention to predict the required embedding.

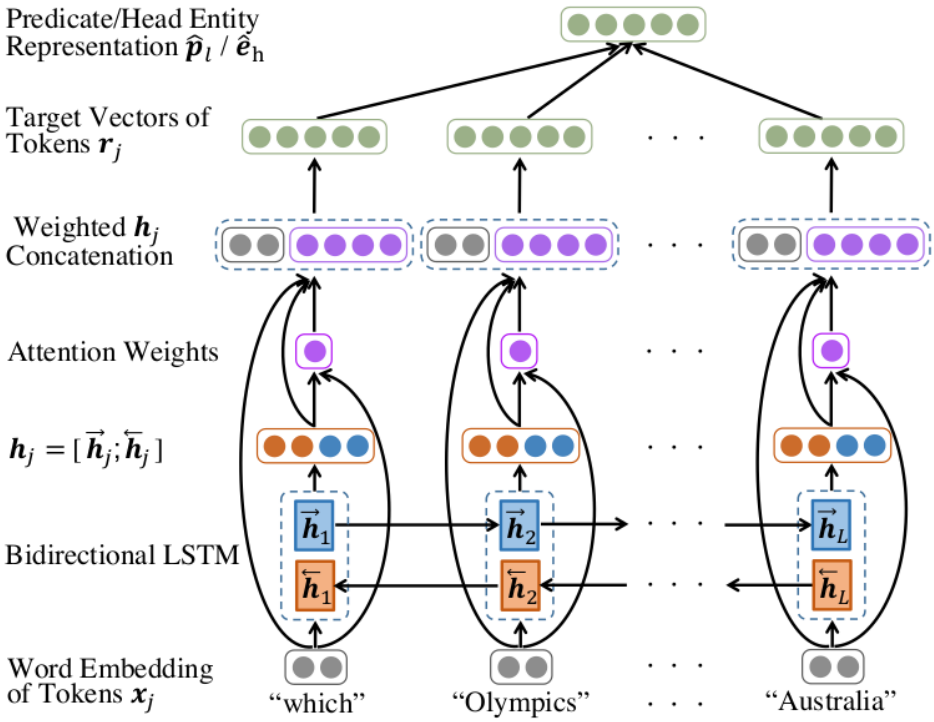


Figure 4.3: Architecture of the Predicate/Head Entity learning model

Figure 4.3 (Huang et. al. 2019) shows the architecture of the model. the layers of the forward-LSTM and backward-LSTM are concatenate, attention weights are calculated, then context vector is computed. It is concatenated with the orignal word embedding (residual connection). The average is taken element wise to get the predicted embeddings.

$$\alpha_j = \frac{\exp(q_j)}{\sum_{i=1}^{L} \exp(q_i)},$$
$$q_j = \tanh(\mathbf{w}^\top [\mathbf{x}_j; \mathbf{h}_j] + b_q)$$

α_j are the attention weights.

$$\hat{\mathbf{p}}_\ell = \frac{1}{L} \sum_{j=1}^{L} \mathbf{r}_j^\top$$

4.3.1 Head Entity Detection Model

There are 15 Million entities in the Freebase knowledge graph. This may lead to a problem. Remember that the last step of the algorithm is to find the fact that is closest to the predicted fact. Search space will be huge in the large knowledge graphs.

A reasonable assumption is used by the authors (Huang et. al. 2019) to solve this problem. Head entity will be from the question itself. This assumption is reasonable, but not every word in the question can be head entity. Head Entity Detection model is used to detect which word in the question can be the head entity.

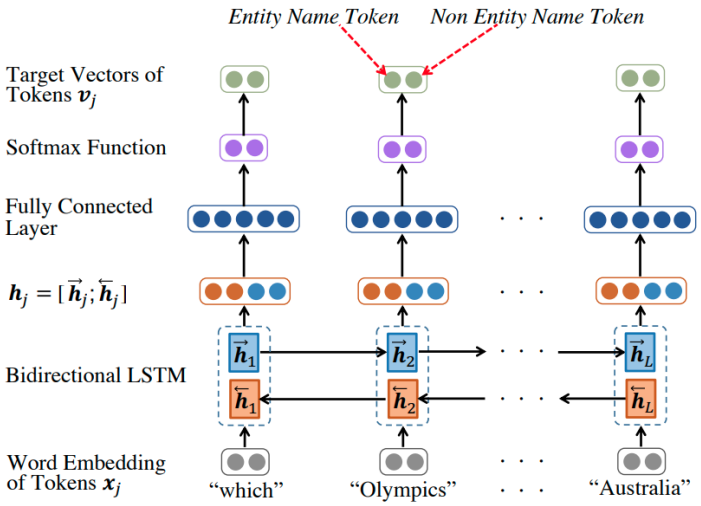


Figure 4.4: Head Detection Model

Figure 4.4 (Huang et. al. 2019) shows the architecture of the head entity detection model. This is the typical bi-directional LSTM architecture with no attention. The last layer consist of two neurons (Indicating whether the token can be Entity name token or not)

4.3.2 Joint Distance Metric

Now we have the predicted head entity, predicate entity and set of all possible head entities. We need to find the fact in the knowledge graph which is closest to the predicted fact. We need a objective function to find that. First set of candidate facts are collected based on the output of the head entity detection model. Let us call the set of candidate fact by C. The objective function used by the author (Huang et. al. 2019) is :

$$\begin{aligned}\underset{(h,\ell,t)\in C}{\text{minimize}} \quad & \|\mathbf{p}_\ell - \hat{\mathbf{p}}_\ell\|_2 + \beta_1\|\mathbf{e}_h - \hat{\mathbf{e}}_h\|_2 + \beta_2\|f(\mathbf{e}_h, \mathbf{p}_\ell) - \hat{\mathbf{e}}_t\|_2 \\ & - \beta_3 sim[n(h), \text{HED}_{\text{entity}}] - \beta_4 sim[n(\ell), \text{HED}_{\text{non}}],\end{aligned}$$

The first three terms finds the fact which are closest to the predicted fact. The last two terms compute the similarity between the two words (one predicted by the head entity model and the other the candidate fact).

Whole algorithm is described in the figure 4.5 (Huang et. al. 2019)

Algorithm 1: The proposed KEQA framework

Input: $\mathcal{G}$, predicates' and entities' names, $\mathbf{P}$, $\mathbf{E}$, Q, a new simple question Q.
Output: head entity h^* and predicate ℓ^*.
/* Training the predicate learning model: */
for Q_i *in* Q **do**
  Take the L tokens of Q_i as the input and its predicate ℓ as the label to train, as shown in Figure 2;
  Update weight matrices $\{\mathbf{W}\}$, $\mathbf{w}$, $\{\mathbf{b}\}$, and b_q to minimize the objective function $\|\mathbf{p}_\ell - \frac{1}{L}\sum_{j=1}^{L}\mathbf{r}_j^\top\|_2$;
/* Training the head entity learning model: */
for Q_i *in* Q **do**
  Take the L tokens of Q_i as the input and its head entity h as the label to train, as shown in Figure 2;
  Update weight matrices and bias terms to minimize the objective function $\|\mathbf{e}_h - \frac{1}{L}\sum_{j=1}^{L}\mathbf{r}_j^\top\|_2$;
/* Training the HED model: */
for Q_i *in* Q **do**
  Take the L tokens of Q_i as the input and its head entity name positions as the label to train;
  Update weight matrices and bias as shown in Figure 3;
/* Question answering processes: */
Input Q into the predicate learning model to learn $\hat{\mathbf{p}}_\ell$;
Input Q into the head entity learning model to learn $\hat{\mathbf{e}}_h$;
Input Q into the HED model to learn $\text{HED}_{\text{entity}}$ and HED_{non};
Find the candidate fact set C from $\mathcal{G}$, based on $\text{HED}_{\text{entity}}$;
For all facts in C, calculate the fact (h^*, ℓ^*, t^*) that minimizes the objective function in Eq. (9).

Figure 4.5: Algorithm KGEQA

4.4 Evaluating Question Answering Systems

Evaluating Question Answering Systems is both important and challenging task. There can be multiple correct answer to a question depending on the context paragraph given to the model as the input. The example is shown in the figure 4.6 (Chen et. al. 2019). Both the answers are correct but the score given by the automatic evaluation is 0, which is wrong.

Context: ...After Peter returns, they eventually figure out her proper care, right down to diaper changes, baths, and feedings. The next day, **two men** (**who are drug dealers**) arrive at the apartment to pick up the package...

Question: Who comes to pick up the package the next day?
Gold Answers: **drug dealers, the drug dealer**
Prediction: **two men**

Human Judgement: 5 out of 5
ROUGE-L: 0
METEOR: 0

Figure 4.6: Example 1

There can also be instance when the automatic evaluation gives a good score to the system but in reality the system is not good. Figure 4.7 (Chen et. al. 2019) shows an example of this situation.

Context: ...David got five exercise tips from his personal trainer, **tip A**, **tip B** ...**Tip A** involves weight lifting, but **tip B** does not involve weight lifting ...
Question: In which tip the skeletal muscle would not be bigger, **tip A** or **tip B**?
Gold Answers: **tip B**
Prediction: **tip A**

Human Judgement: 1 out of 5
F1: 0.66

Figure 4.7: Example 2

In this example we can easily see that the answer given by QA model is wrong. Yet the score given by the automatic evaluation system (F1) is 0.66. These tells us that there is a need for the good automatic evaluation. Next section **??** discusses the transformer based approach on the automatic evaluation of the QA systems.

4.4.1 AVA: an Automatic eValuation Approach to Question Answering Systems

AVA (Vu et. al. 2020) is the transformer based approach. Bascially a transformer model is made using RoBERTa which takes the subset of question, reference answer, candidate

answer as the input and gives the binary output, depending on semantic similarity of the reference and the candidate answers. Authors proposed three type of models. The models are A_0 : **Text-Pair Embedding**, A_1 : **Improved Text-Triple Embedding**, A_2 : **Peer-Attention for the pair of Transformer based models**. These are discussed in the next sections.

A_0 : Text-Pair Embedding

We have (q, r, t), the question, reference text and the t is the candidate answer. Text-Pair embedding is very simple. There are 3 combination of the pair (q, r), (r, t), (q, t). We have three models depending on which pair is given as an input to the RoBERTa model. RoBERTa model gives us the representation of the text pair. Using this representation we can easily make a binary classifier.

Along with these 3 models, there is one more model. The concatenation of the representation of these three pair. In total we have four models for the A_0 configuration.

A_1 : Improved Text-Triple Embedding

There is an obvious limitation in A_0 configuration. The model is limited by the pair, for example (q, r) configuration has no relation to the candidate answer. A_1 configuration uses the whole triplet. We can only give a pair as an input to the RoBERTa model. We use the concatenation operator$o$ between a pair, now we have two sentence to give as an input to the RoBERTa model. We have the representation of the triplet. Now we can easily use the binary classification to find out whether candidate answer answers the question or not.

Similar to A_0 configuration there can be four models. $\{(q, r\, o\, t)$, $(r, q\, o\, t)$, $(t, q\, o\, r)\}$ and the concatenation of these three pairs representation.

A_2 : Peer-Attention for the pair of Transformer based models

There is a strong correlation/connection between the pairs in the set $\{(q, r\, o\, t)$, $(r, q\, o\, t)$, $(t, q\, o\, r)\}$. A_1 configuration is not representing that. A_2 configuration introduces the pair attention mechanism using which different pairs can attend to each other. Let us assume that two pairs are (a, a') and (g, g') the pair attention is similar to encoder decoder model in the machine translation networks.

Figure 4.8 (Vu et. al. 2020) shows the peer attention network. This is very similar to encoder decoder model. It works on two phases, first phase is encoding, both pair of sentences are encoded using the RoBERTa model. In the second phase the output of the first phase is given as the input to the decoder of the opposite sentence. These four representation are then given to the classification layer. Thus we have model which uses peer attention.

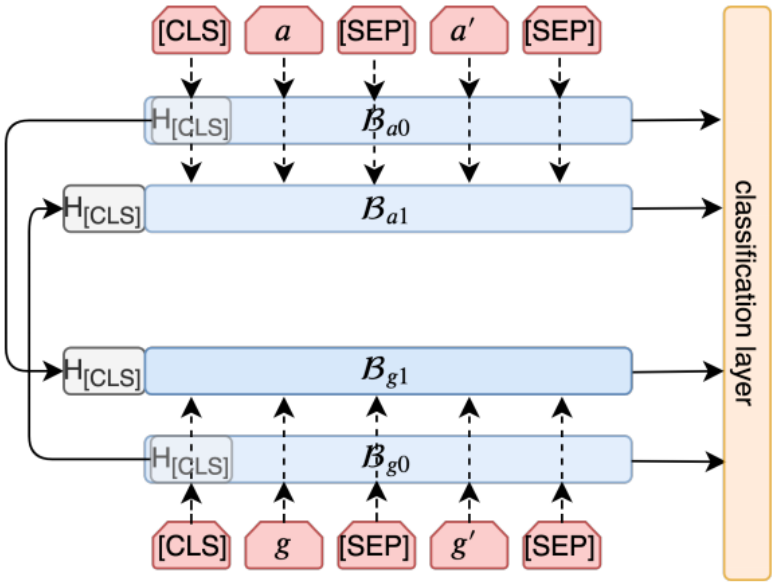


Figure 4.8: Peer Attention Network

Datasets

The datasets used for the evaluation purposes are AVA-NQ and AVA-GPD. Figure 4.10 (Vu et. al. 2020) shows the statistics of the AVA-NQ Dataset. Positive and negative pairs are created from the AS2-NQ dataset (for more details see Vu et. al. 2020).

	AS2-NQ			**AS2-NQ Qs with multiple As**			**AVA-NQ**		
data split	#Qs	#As	#wrong-As	#Qs	#As	#wrong-As	positives	negatives	total
NQ-dev	4,263	134,691	1,320,812	1,478	3,376	64,187	11,556	206,497	218,053
NQ-train	105,020	10,288	33,294,803	2,360	6,392	96,152	26,100	432,913	459,013

Figure 4.9: Statistics of AVA-NQ Dataset

Figure 4.9 (Vu et. al. 2020) shows the statistics of tha AVA-GPD dataset. Similar to AVA-NQ, AVA-GPD is created using the AS2-GPD dataset (for more details see Vu et. al. 2020). It also contains both positive and negative pairs of reference and the candidate answers.

	AS2-GPD			**AS2-GPD Qs with multiple As**			**AVA-GPD**		
data split	#Qs	#As	#wrong-As	#Qs	#As	#wrong-As	positives	negatives	total
GPD-train	262	5,399	20,801	245	5,382	20,748	183,894	349,765	533,659
GPD-dev	283	8,682	19,618	276	8,674	19,502	430,230	426,246	856,476
GPD-test	294	9,412	19,988	281	9,399	19,790	479,028	449,625	928,653

Figure 4.10: Statistics of AVA-GPD Dataset

Evaluation Results

Figure 4.11 (Vu et. al. 2020) shows the evaluation result on the AVA-NQ and the AVA-GPD datasets. F1 score is used as a metric. We can see that $A_0((q,r))$ performed very poorly as expected as there is no candidate answer in the pair. The best performance is given by the A_2 (peer attention) configuration with the F1 of 0.7472.

training set from development set from	AVA-NQ AVA-NQ	AVA-GPD AVA-GPD
Model	**F1 on AVA-GPD-Test**	
Linear Classifier	0.0000	0.3999
$\mathcal{A}_0(\{(q,r)\})$	0.0004	0.0695
$\mathcal{A}_0(\{(r,t)\})$	0.3778	0.6247
$\mathcal{A}_0(\{(q,t)\})$	0.5801	0.6713
$\mathcal{A}_0(\mathcal{D}_0)$	0.3962	**0.6807**
$\mathcal{A}_1(\{(q,r\circ t)\})$	0.3788	0.7014
$\mathcal{A}_1(\{(r,q\circ t)\})$	0.4583	0.7383
$\mathcal{A}_1(\{(t,q\circ r)\}))$	0.4517	0.7236
$\mathcal{A}_1(\{(q,r\circ t),(t,q\circ r)\})$	0.3546	0.7421
$\mathcal{A}_1(\{(r,q\circ t),(t,q\circ r)\})$	0.4002	**0.7447**
$\mathcal{A}_1(\{(r,q\circ t),(q,r\circ t)\})$	0.4873	0.7435
$\mathcal{A}_1(\mathcal{D}_1)$	0.4121	0.7303
$\mathcal{A}_2((r,q\circ t),(t,q\circ r))$	0.4187	**0.7472**

Figure 4.11: Evaluation Result

A_1 configuration also performed comparably to A_2 configuration. A_2 needs more time for the training because of the complex network used by A_2 configuration. There is a trade off between the performance and the resources. A_2 requires more resources but give best performance among A_0, A_1 and A_2. While A_1 requires less resource than A_2, still gives the comparable performance.

4.5 Summary

In this chapter we discussed one of the important application of the knowledge graph embeddings (Question Answering). We discussed a model that can answer the simple questions using the knowledge graph. We discussed the architecture, important components of the model. Predicate Learning Model which uses Bi-LSTM architecture with attention, which predicts the predicate/relation. We also discussed Head Entity Detection model which reduces the search space by assuming that head entity is the part of the question. Joint Distance Metric is used to select the fact which is closest to the predicted fact to get the answer to the question.

Apart from this model we discussed challenges in automatic evaluation of the question answering system. We discussed a transformer based approach to evaluate the question answering system. We discussed three configuration of the AVA model. We also discussed which configuration is best out of all three.

Chapter 5

Question Answering using Deep Learning

Deep Learning models are very important in the field of NLP. In this chapter we will focus on those models. First introduction is given in section 5.1. Deep learning models are data-driven. Some common datasets are described in section 5.2. SQuAD, is the most famous dataset for Question Answering task, it is described in section 5.2.1. MultiRC (which contains multiple choice questions) is described in section 5.2.2. Natural Questions (NQ) dataset which is used for open-domain question answering is described in section 5.2.3. MS MARCO is described at section 5.2.4 and NEWSQA which contains paragraphs from news article is described in section 5.2.5. Two important deep learning models for question answering are discussed. BiDAF is described in section 5.3. BERT for Reading Comprehension is described in section 5.4. The chapter is summarized in section 5.6.

5.1 Introduction

Question Answering is one of the important downstream task in NLP. User asks the question to the Question Answering system in the natural language. The goal of the system is to give the answer to the user in the natural language. Broadly speaking there are two types of question answering task, one is reading comprehension, and the other is open-domain question answering. In the reading comprehension the input to the question answering system is the paragraph/context and the question, the output is the answer. In the open-domain question question answering, the input is several documents and the question, the output is the answer.

In this chapter we will discuss the Reading Comprehension part, more specifically we will discuss the question answering systems with some constraint. The answer to the question is in the span to the paragraph. This is the strong constraint on the system. As we all know that this is not always the case often we need to infer answers from the different part of the passages.

5.2 Some Common Datasets

In this section we will discuss some common datasets, which are used for training the QA systems. The most popular Dataset for the QA system is SQuAD. SQuAD perfectly fits the problem which we are going to discuss in this chapter. Some of the important datasets except SQuAD are MultiRC, Natural Questions, MS MARCO, NEWSQA. Which we will discuss in the next subsections.

5.2.1 SQuAD

SQuAD is one of the most popular datasets for the Question Answering task. SQuAD (CS224N Standford Slides) contains passages, the questions and the answers. The answers are from the span of the passages. SQuAD 2.0 contains more than 100K questions. There are some questions which have no answers. Models have to predict this also.

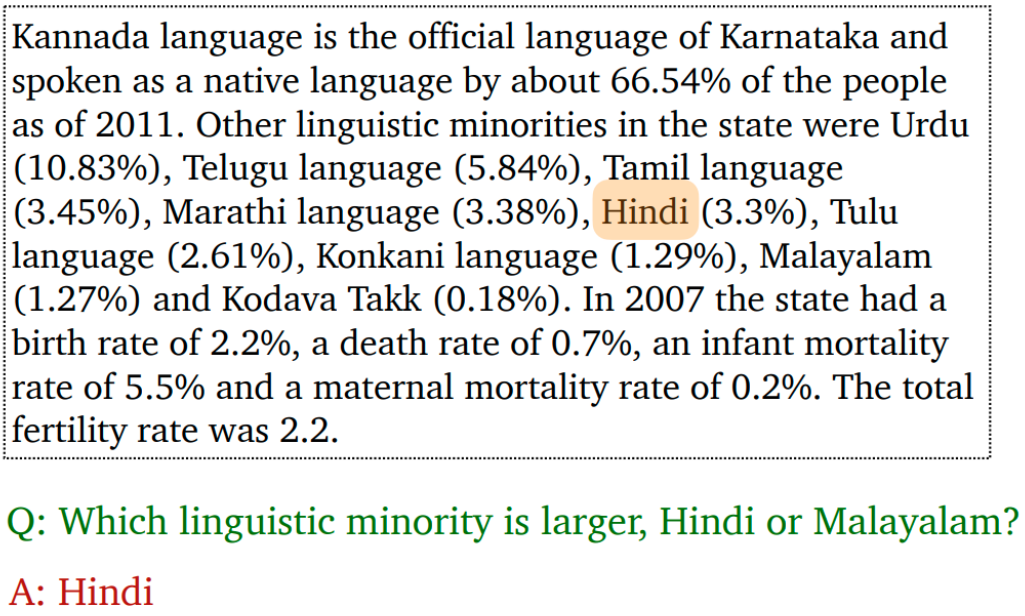


Figure 5.1: Example of SQuAD

Figure 5.1 (CS224N Standford Slides) shows an example of the SQuAD dataset. Dataset have a paragraph, a question and the answer is the span of the paragraph. We can see how models are performing at https://rajpurkar.github.io/SQuAD-explorer/. We can see that the transformer based models have outperformed human.

5.2.2 MultiRC

MultiRC (Khashabi et al. 2018) contains questions which can be answerd only by collecting information from different parts of a paragraph. The answer is no longer the span of the paragraph. Every questions have options and there can be more than one correct answer. The models cannot cheat by selecting the best possible answer. It have to look into every option and determine whether it is correct answer or not.

S3: Hearing noises in the garage, Mary Murdock finds a bleeding man, mangled and impaled on her jeep's bumper.
S5: Panicked, she hits him with a golf club.
S10: Later the news reveals the missing man is kindergarten teacher, Timothy Emser.
S12: It transpires that Rick, her boyfriend, gets involved in the cover up and goes to retrieve incriminatory evidence off the corpse, but is killed, replaced in Emser's grave.
S13: It becomes clear Emser survived.
S15: He stalks Mary many ways.

Who is stalking Mary?

A)* Timothy	D) Rick
B) Timothy's girlfriend	E) Murdock
C)* The man she hit	F) Her Boyfriend

S1: Most young mammals, including humans, play.
S2: Play is how they learn the skills that they will need as adults.
S6: Big cats also play.
S8: At the same time, they also practice their hunting skills.
S11: Human children learn by playing as well.
S12: For example, playing games and sports can help them learn to follow rules.
S13: They also learn to work together.

What do human children learn by playing games and sports?
A)* They learn to follow rules and work together
B) hunting skills
C)* skills that they will need as adult

Figure 5.2: Example Of MultiRC

Figure 5.2 (Khashabi et al. 2018) shows the example of the MultiRC dataset. We can easily see that, to answer the question model have to look into multiple part of the paragraph (which is not there in SQuAD dataset).

5.2.3 Natural Questions (NQ)

Natural Questions dataset is designed by google. It contains natural queried and answer to them is in a document (not just a paragraph)/. This dataset is for open-domain question answering. It contains both short and long answer. It contains almost 307K training examples, 8K examples for development, and 8K examples for testing.

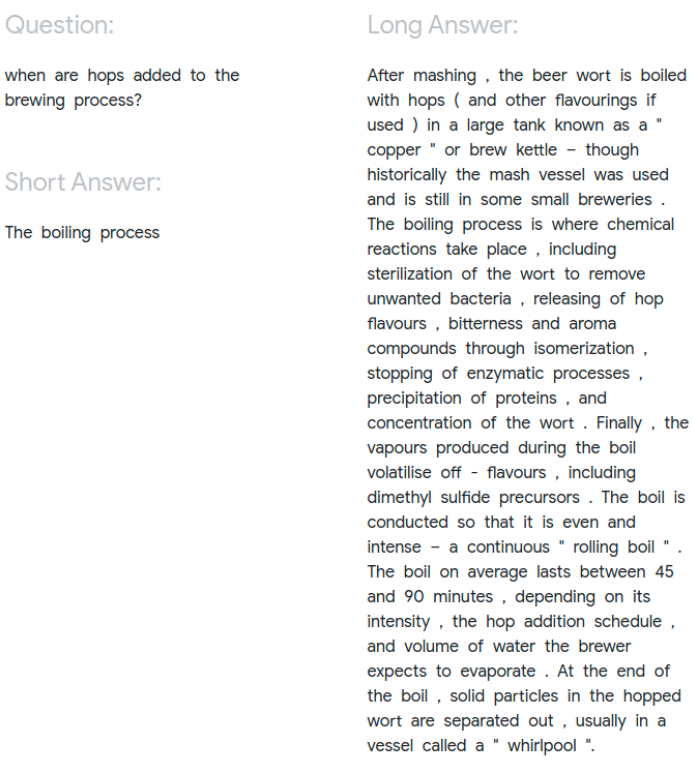


Figure 5.3: Example Of NQ

Figure 5.3 shows the example of the NQ dataset, we can see that it contains both short and long answers. Both answer are from the wikipedia.

5.2.4 MS MARCO

MS MARCO dataset is constructed by the researchers at microsoft. This dataset can be used for QA, Ranking, Keyphrase extraction, and many other tasks. Dataset contains the queries asked by the user on the bing (search engine). The answers are human generated (not a span in the paragraph). It also contains well formed answers, which can be used for natural language generation task. It contains more than 1 Million questions.


```
{
    "answers":["A corporation is a company or group of people authorized to act as a single entity and recognized as such in law."],
    "passages":[
        {
            "is_selected":0,
            "url":"http:\/\/www.wisegeek.com\/what-is-a-corporation.htm",
            "passage_text":"A company is incorporated in a specific nation, often within the bounds of a smaller subset of that nation,
such as a state or province. The corporation is then governed by the laws of incorporation in that state. A corporation may issue
stock, either private or public, or may be classified as a non-stock corporation. If stock is issued, the corporation will usually be
governed by its shareholders, either directly or indirectly."},
        ...
        }],
    "query":". what is a corporation?",
    "query_id":1102432,
    "query_type":"DESCRIPTION",
    "wellFormedAnswers":"[]"
}
```


Figure 5.4: Example Of MS MARCO

Figure 5.5 shows an example of the MS MARCO dataset. We can see that it contains answers, set of passages, query etc.

5.2.5 NEWSQA

NEWSQA is a dataset constructed from the news (CNN) data. It is a challenging machine comprehension dataset. It contains around 100,000 human generated question-answer pairs.

Reasoning	Example	Proportion (%) *NewsQA*	*SQuAD*
Word Matching	Q: **When were** the **findings published**? S: Both sets of research **findings were published Thursday**...	32.7	39.8
Paraphrasing	Q: **Who** is the **struggle between** in Rwanda? S: The **struggle pits ethnic Tutsis**, supported by Rwanda, **against ethnic Hutu**, backed by Congo.	27.0	34.3
Inference	Q: **Who** drew **inspiration** from **presidents**? S: **Rudy Ruiz** says the lives of US **presidents** can make them **positive role models** for students.	13.2	8.6
Synthesis	Q: **Where** is **Brittanee Drexel** from? S: The mother of a 17-year-old **Rochester**, **New York** high school student ... says she did not give her daughter permission to go on the trip. **Brittanee** Marie **Drexel**'s mom says...	20.7	11.9
Ambiguous/Insufficient	Q: **Whose mother** is **moving** to the White House? S: ... **Barack Obama's mother-in-law**, Marian Robinson, will **join** the Obamas at the **family's private quarters** at 1600 Pennsylvania Avenue. [Michelle is never mentioned]	6.4	5.4

Figure 5.5: Example Of NEWSQA

Figure Trischler et al. 2017 gives some examples of NEWSQA dataset. Some questions requires general knowledge for answering. Some also requires inferencing to answer the question.

5.3 BiDAF: The Bidirectional Attention Flow Model

BiDAF is LSTM and attention based model, which uses two different attention (Context to Query, Query to Context).

Figure 5.6 (Seo et al. 2017) shows the architecture of BiDAF. First step is encoding/embedding (which is the first step of almost all the NLP applications). BiDAF uses both character embedding and the word embedding layer. The second and the most important step is the attention step. As the name suggests BiDAF uses two type of attention, one context to query attention, and query to context attention. The output of the attention is given to two softmax layer which predicts the start and the end token in the paragraph for the question.

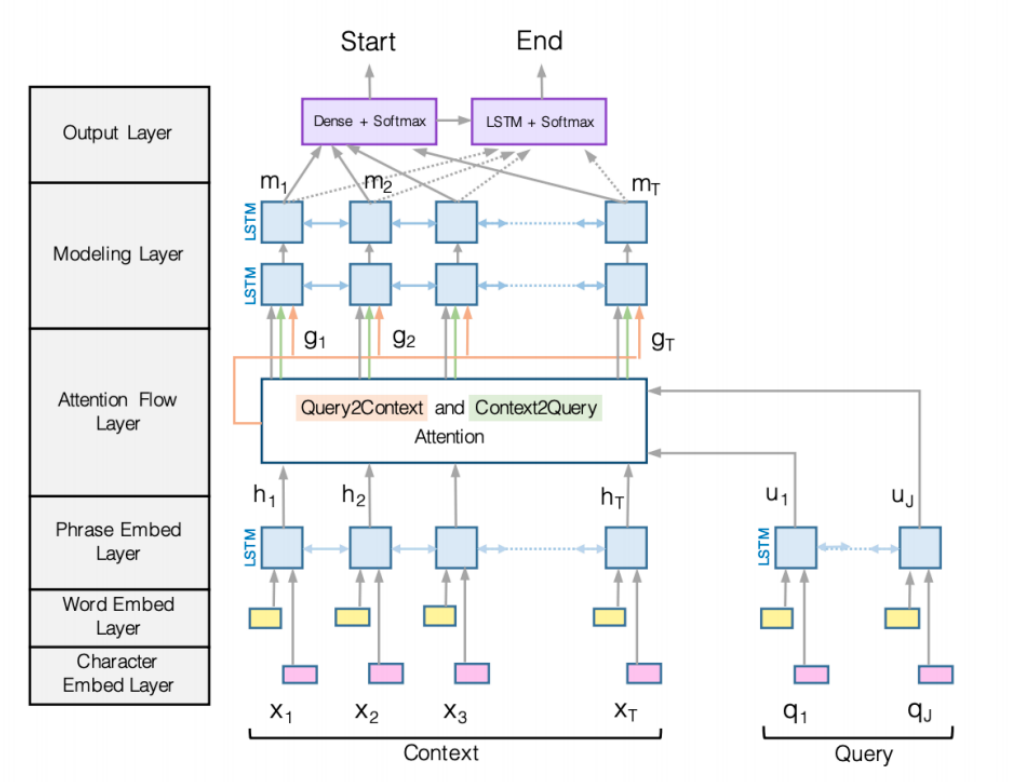


Figure 5.6: BiDAF Architecture

5.3.1 Encoding

First step is encoding, character embedding and word embedding are given as a input to the LSTM layer for phrase embedding.

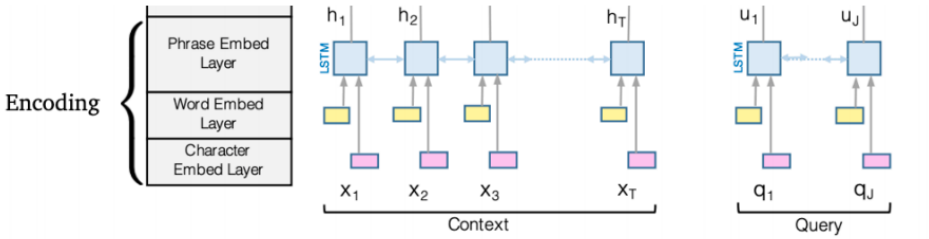


Figure 5.7: BiDAF Encoding Layer (CS224N Standford Slides)

This is done for both the context and the question.

$$e(c_i) = f([GloVe(c_i); charEmb(c_i)])$$

$$e(q_i) = f([GloVe(q_i); charEmb(q_i)])$$

After this step we have a representation for the context as well as for the question.

5.3.2 Attention

BiDAF uses context to query as well as query to context attention. Context to query attention calculates attention for each context word. How much a context word should give attention to the query words, this is answered by context to query attention.

Query to context attention is for the query, it calculates which context words are most relevant to the query. We have m context to query attention vector (context vector) (m is the number of context words). We have one query to context vectors. To calculate both attention we first need similarity score between context words and the question words.

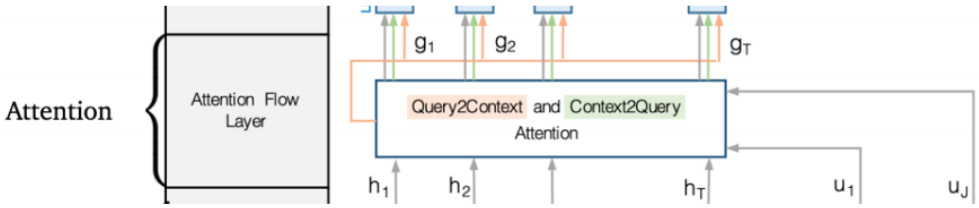


Figure 5.8: BiDAF Attention (CS224N Standford Slides)

$$
\begin{aligned}
S_{i,j} &= \mathbf{w}^{\mathbf{T}}{}_{sim}\left[\mathbf{c}_i; \mathbf{q}_j; \mathbf{c}_i \odot \mathbf{q}_j\right] \in \mathbb{R} \quad \odot \; is\; Hardmard\; Product \\
\alpha_{i,j} &= softmax_j(S_{i,j}) \in \mathbb{R} \\
\mathbf{a}_i &= \sum_{j=1}^{M} \alpha_{i,j}\mathbf{q}_j \in \mathbb{R}^{2H} \qquad Context\; to\; query\; Attention \\
\beta_i &= softmax_i\left(\max_{j=1}^{M}\left(S_{i,j}\right)\right) \in \mathbb{R}^N \\
\mathbf{b} &= \sum_{i=1}^{N} \beta_i \mathbf{c}_i \quad Query\; to\; Context\; Attention \\
g_i &= \left[\mathbf{c}_i; \mathbf{a}_i; \mathbf{c}_i \odot \mathbf{a}_i; \mathbf{c}_i \odot \mathbf{b}\right] \quad Final\; output\; of\; the\; attention
\end{aligned}
$$

5.3.3 Modeling and output layers

After the attention layer, the final layers are modeling and the output layer. Again BiLSTM is used to model interaction between the output of the attention layers.

$$\mathbf{m_i} = BiLSTM\left(g_i\right)$$

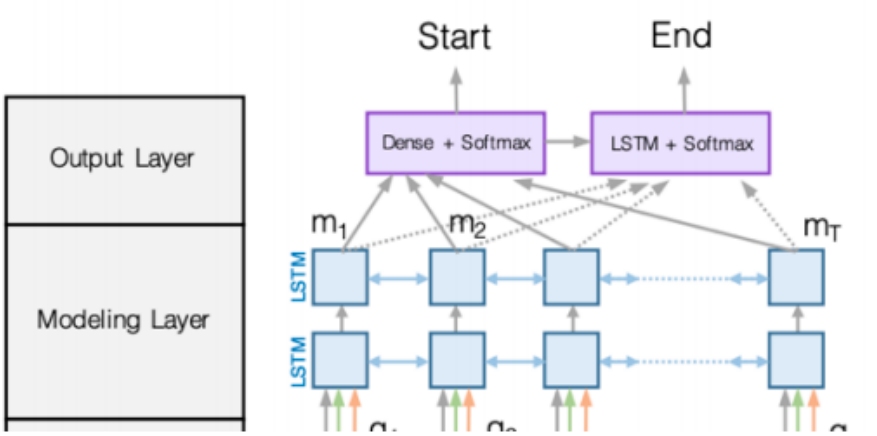


Figure 5.9: BiDAF modeling and output layer

Now two classifiers are used to predict the start and the end of the span.

$$p_{start} = softmax\left(\mathbf{w}^{\mathbf{T}}{}_{start}\left[\mathbf{g}_i; \mathbf{m}_i\right]\right)$$
$$p_{end} = softmax\left(\mathbf{w}^{\mathbf{T}}{}_{end}\left[\mathbf{g}_i; \mathbf{m}^{'}{}_i\right]\right)$$
$$\mathbf{m_i^{'}} = BiLSTM(\mathbf{m_i})$$

5.4 BERT for Reading Comprehension

BERT is the Langauge model which uses Transformer's Encoder. BERT uses MLM objective and Next sentence prediction for training. BERT can be used for many tasks, one of them is Reading Comprehension.

We discussed LSTM and attention based approach (BiDAF), this is transformer based approach. We finetune BERT based model on a particular task. Pretraining is the game changer, it helps significantly in improving the model performance.

5.4.1 Encoding

Similar to the BiDAF model, the first step in the BERT based model is encoding

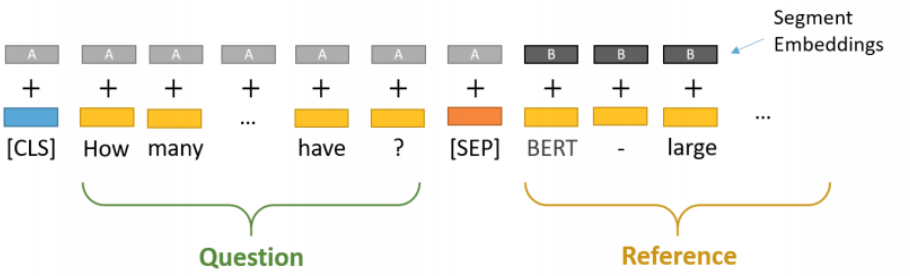


Figure 5.10: BERT Encoding

Figure 5.10 (https://mccormickml.com/page2/) shows the encoding. Questions and Reference are given to the BERT model, with segment encoding, position encoding (similar to the transformer) and the word embedding.

5.4.2 Output Layers

After encoding, we just need to build the output layer (self attention is similar to the transformer). Output layer will predict the start and the end token in the context/passage.

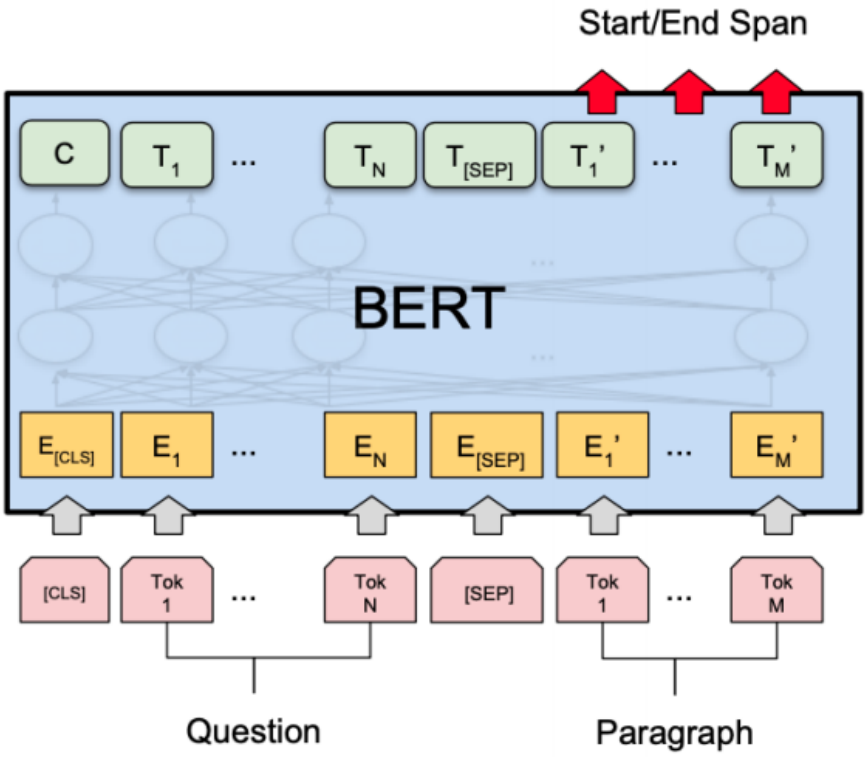


Figure 5.11: BERT for Reading Comprehension

$$p_{start} = softmax\left(\mathbf{w^T}_{start}\mathbf{H}\right)$$
$$p_{end} = softmax\left(\mathbf{w^T}_{end}\mathbf{H}\right)$$

Where, $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_N]$ are hidden vectors of the context, returned by the BERT

5.4.3 Span BERT

Span BERT is very similar to the BERT. It differs from the BERT only in Pre-Training objective. BERT replaces 15% of the token with the MASK token. Span BERT replaces whole Span with the MASK token rest of the details are same for Span BERT and BERT. Span BERT performs better than the BERT.

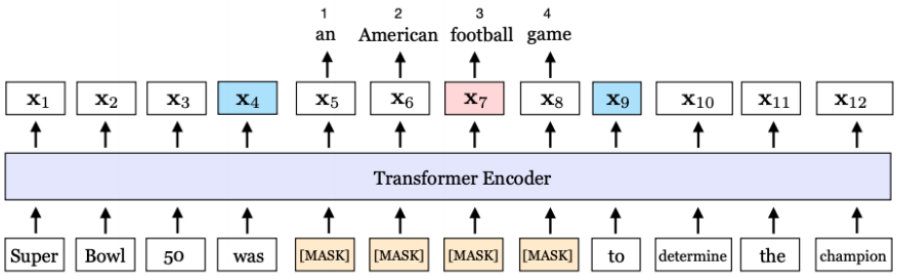


Figure 5.12: Span BERT pre-training

5.5 Comparision between BiDAF and BERT

This section gives a small comparision between BiDAF and BERT. BiDAF is LSTM and attention based model, BERT is transformer based model. BERT have much more parameters then BiDAF.

They have so many differences but they are fundamentally not so different. BiDAF uses context to query and query to context vector, while BERT uses self attention between query and query, query and context, context and query, query and query, infact we can improve performance of BiDAF by adding the attention between query to query in BiDAF.

5.6 Summary

Question Answering is the task in which user asks a question and system gives the answer to the question. There are two types of question answering task, close domain (Reading Comprehension) and open-domain question answering. In this chapter we discussed about reading comprehension task. We discussed some common datasets used for training the QA systems. SQuAD is the most popular dataset. Answer to the question in the SQuAD is in the span of the paragraph (which is one of the limitation of SQuAD model). Next we discussed various others datasets, MultiRC which contains questions that requires multiple sentence to answer the question. It contains multiple choice questions. Natural Questions (NQ) is a dataset for close-domain question answering systems, it contains questions, wikipedia links, short and long answers. After discussing some common datasets, we discussed BiDAF model which uses the LSTM with attention, it involves encoding step (which uses character embedding, word embedding and phrase embedding layer), Context to Query attention, Query to Context attention, Modeling and output layer. Next we discussed BERT for reading comprehension. We fine tune BERT language model with additional two output layers, one for predicting start token, other for predicting end token. We then discusses Span BERT which uses different pre-training objective then BERT. Instead of masking 15% of the token. Span BERT masks span of the sentence. Span BERT performs better than BERT in the Question Answering task. We then compared BiDAF and BERT model. Which have many differences but fundamentally they are not so different, BERT uses self attention between context and query, query and context, context and context, query and query, while BiDAF uses context to query attention, and query to context attention.

Chapter 6

Joint IIT LG Project (Question Answering Using Knowledge Graph)

This chapter contains the brief information about the Joint IIT LG Project. The problem statement is discussed in section 6.1. Almost every big project have a pipeline, which is described in section 6.2. One of the component of the pipeline in Question Classifier, which is described in section 6.3. After extracting some information from the knowledge graph a pre-trainied QA system is required, which is described in section 6.4. After the extraction of the answer natural language generation is required, which is discussed in section 6.5. Some of the answers can be very long so summarization module is disccused in section 6.6. UI for debudding/testing is described in section 6.7. The chapter is summarized in section **??**.

6.1 Problem Statement

The problem statement is very common, the task is to build the Question Answering system which uses the knowledge graph. User can ask any question related to any L.G. product. The Model have to give the answer to the question from the product manual.

The knowledge graph is used to store the information of the product manual in the structured manner so that answering questions from the product manual becomes relatively easy.

6.2 Pipeline

Almost every big projects have a pipeline which gives the high level overview of the solution proposed to the problem. Figure 6.1 shows the pipeline. Most part of the pipeline is fairly intuitive. Knowledge graph is a graph so it contains the nodes and the edges. Usually the nodes are the entities and the edges are the relations, but in this project the nodes are not entities, nodes contains sentences, edges are the relations.

Top 50 nodes are extracted from the knowledge graph based on the textual similarity features. Now from that 50 nodes, top 10 nodes are selected using the vector similarity and the textual similarity. It is assumed that the answer is in these 10 nodes. Question

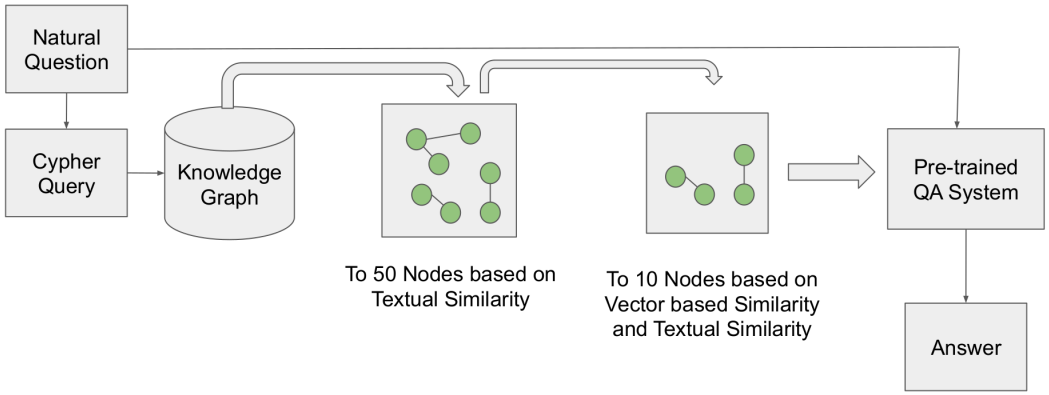


Figure 6.1: Pipeline

and the contents of top 10 nodes are given to pre-trained QA system (which also contain the classifier which is described in the next section) to get the required answer.

6.3 Question Classifier

There can be different types of questions (Factoid, Non-Factoid, Boolean,List) and approach to get the answer depends on the type of the question.

- **Factoid Type Questions :** These are fact-based questions. Usually factoid type questions start with 'wh' words. For example what is the capital of India? The answers are generally named entities.
- **List Type Questions :** These are the questions in which a list is required to answer them for example List the name of 10 comedy movies? The answer to this question will be a list of the named entities. Another example can be: List the steps to write a good report? The answer to this will not be a list of the named entities instead it will contain the list of statements in some order.
- **Boolean [yes or no] :** These are the questions in which the answer is a boolean (either yes or no). An example of confirmation-type questions is: Does the sun rises in the east? The answer is simply yes or no.
- **Non Factoid Type Questions :** These are open-ended questions that require complex answers to answer them. These can be opinions, descriptions, explanations. An example of non-factoid questions is How to read research papers? The answer to this question will contain some opinion and the answer will be descriptive.

The Bert based sequence classifier is fine-tuned to classify any question in these four categories. Figure 6.2 (Devlin et. al. 2018) shows the BERT based sentence classifier.

6.4 Pre-trained QA Systems

Pre-trained QA Systems is required after the node extraction step. Top 10 nodes are extracted from the knowledge graph. It is assumed that the answer to the question is in the contents of the top 10 nodes extracted. We have the context paragraph (Top 10

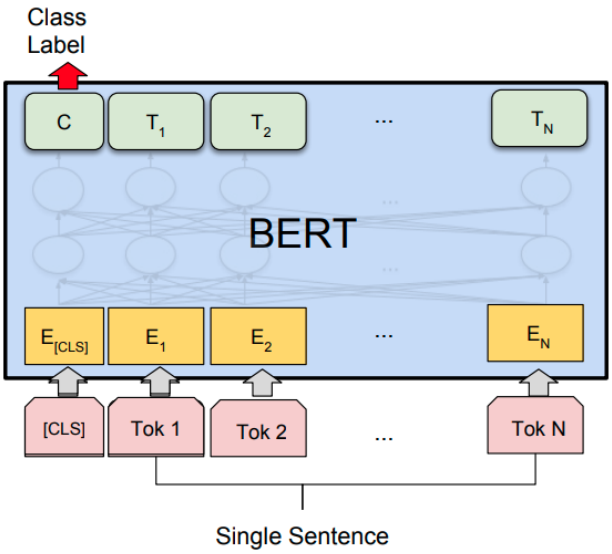


Figure 6.2: BERT based sentence classifier

nodes), we also have the question asked from the user and the result of the question classifier.

Depending on the result from the classifier, different models are used. If the result from the classifier is that the question is factoid, then the BERT based model is used to extract the answer from the context paragraph.

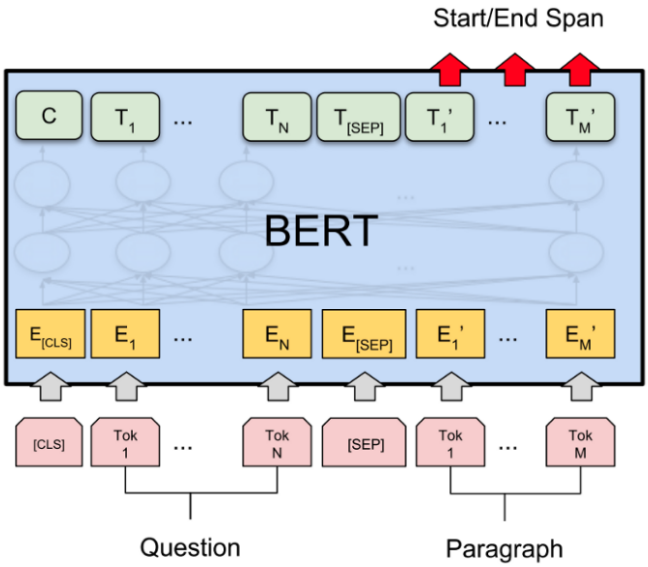


Figure 6.3: BERT Question Answering Task

Figure 6.3 (Devlin et. al. 2018) shows the BERT model for the Question Answering task. Question and context paragraphs are given as an input to the BERT. We get the start and the end span token as the output (thus the answer).

If the result from the classifier is that the question is non-factoid, then T5 model is used

for answering that questions.

6.5 Natural Language Generation

Natural Language Generation is also a very important task in the NLP. It is also important for this kind of project, which requires giving extracted answer to the user. Let us take an example, what if we ask a question from two models "Is Delhi the capital of India?". One model gives the answer "yes" and the other model gives the answer "yes, the Delhi is the capital of India.". We can easily see that the answer given by the second model is better than the answer given by the first model. The difference between these two models is that the second model is updating the answer with some text.

A rule based natural language generation module is used to generate the complete answer so that user can have good experience by interacting with the chat bot.

6.6 Summarization

The answers which model gives may be very long. Long answers (typically $>$ 40 words) cannot be given to the user (for good experience with the chat bot). The summarization module is used to summarize the long answers.

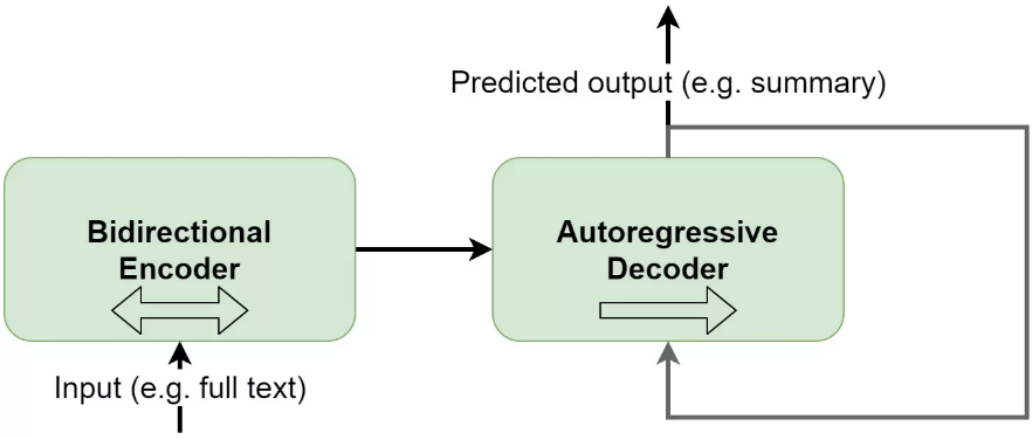


Figure 6.4: BART for Text Summarization

BART (Lewis et. al. 2019) is used for this purpose. Figure 6.4 (Machine Curve Blog). shows the text summarization performed by the BART model.

6.7 UI for Debugging/Testing/Understanding

There can be different ways to debug/test the model. Along with my seminar, i also made a simple UI for testing/debugging purposes. UI for natural language generation part contains a field to enter the question, answers. Module for nature language generation will generate the output based on the question and the answer. The UI will then show the answer to the user.

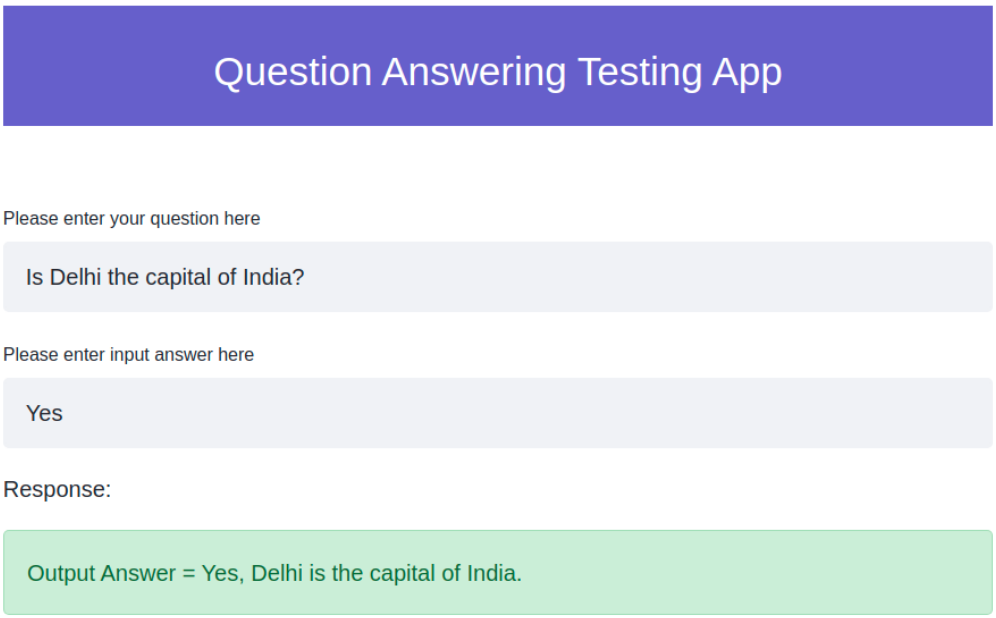


Figure 6.5: UI for testing Natural Language Generation

Figure 6.5 shows the UI. UI is made using the streamlit library, which is very easy to use.

6.8 Summary

We first discussed the problem statement, which is to build the question answering system using the knowledge graph. Next we discussed the pipeline of the project. The main components in the pipline are the knowlege graph, Question Classifier and, Pre-trained QA systems. Question Classifier used in the project is BERT based sequence classifier. Pre-trained QA systems are used based on the result of the Question Classifier. BERT is used for answering factoid based questions. T5 is used for answering the non-factoid questions. A brief information is given about the natural language generation module which is rule based. If the answers are long (>40 words), then the summarization module is used to summarize the answer. UI for debugging/Testing is also described, which is made using the streamlit library.

Chapter 7

Summary, Conclusion and Future Work

This is the last chapter of the report. The report is summarized in section 7.1. The conclusion of the report is given in the section 7.2. Finally the future work is given in the section 7.3.

7.1 Summary

In the report We first discussed various knowledge graphs which we can be used in NLP applications. We also discussed various tools through which we can apply the knowledge graph in our applications. We first discussed WordNet, which is a lexical database. We discussed various relationship that are between the words in the WordNet (*synonymy*, *antonymy*, *hyponymy*, *troponymy*, *entailment*). We then discussed the tools by which we can access the WordNet (nltk library and web). We discussed BabelNet, which is a very large multilingual semantix network. It is constructed using the data from both WordNet as well as Wikipages. There is a mapping algorithm which matches words from the wikipedia to the sense in the WordNet. Similiar to WordNet we then discussed the tools for using the BabelNet (using the Babelnet API and the Web). ConceptNet is also a semantic network which is based on the Open Mind Common Sense database, which contains facts like *A doctor is highly trained professional.* ConceptNet converts such facts which are in unstructed format to the semantic network which is in structured format. Similar to BabelNet we then discussed the API to use ConceptNet and the web interface to access the ConceptNet. Fourth knowledge base which we discussed was Yago, which combines two sources into one (Wikidata and Schema.org (and its siblings)). We also discussed some tools to access the Yago (both using SparQL and the web interface). Last knowledge base we discussed was Freebase which is very large knowledge base, but google discontinued the freebase on 2 May 2016. We can still download the dump of 2016 and use the freebase in our applications.

In the third chapter, we first discussed the intuition behind the knowledge graph embedding algorithms. The intuition is that the nodes having similar neighbours should have similar knowledge graph embeddings. We also discussed the importance of knowledge graph embeddings in the applications like link prediction, question answering. Next we disscused the translation based models, which uses vector addition (translation) between

the head entity embeddings (or projection of head entity) and the relation embeddings to get the embeddings of the tail entity. In the translation based models we first discussed the TransE model which uses simple vector addition (translation) between the head entity embedding and the relation embedding to get the tail entity embedding ($\mathbf{h} + \mathbf{r} = \mathbf{t}$). Next we discussed TransH model, which projects the head and tail entity into the hyper-plane specific to the relations, then the translation is performed similar to the TransE model. TransR is very similar to TransH, instead of projecting the embeddings to the hyper-plane, TransR proposes that the entities and the relations should have different spaces. First both the head and tail entity are projected into different space specific to the relation and then the translation is performed similar to the TransE model.

Next we discussed the compositional based models, which uses the compositional operator between the head enitity and the tail entity. we discussed RESCAL which uses tensor product as a compositional operator between the head and the tail entity. HoLE uses the circular correlation operator between the head and the tail entity. RESCAL algorithm have more trainable parameters (because of the tensor product), which makes the algorithm not scalable. HoLE have comparatively less number of parameter (comparable to TransE), so this makes HoLE algorithm more scalable. We also discussed different compositional operator that can be used. We discussed the concatenation, projection and non-linearity as the compositional operator, this have limitation of less interaction among the embeddings compared to the tensor product.

In the third part of the third chapter, we discussed the neural network based models. We first discussed SLM model, which have very less number of interaction between the embeddings which resulted in the poor performance of the algorithm. We also discussed NTN model, which increases the interaction between the embeddings but increases the number of trainable parameter. This results that the algorithm is less scalable. Next we discussed the convolution network based knowledge graph embeddings. First we discussed Conv2d algorithm, which concatenates the entities embeddings (head entity and tail entity) in 2 dimension (to increase the interaction), then normal convolution architecture was used. ConvKB uses the similar idea but gets very good result, instead of concatenating only the entities embeddings. ConvKB concatenates all the three embeddings(head entity, relation,tail entity).

In the last part of the third chapter, discussion was focused on the OpenKG. First we discussed the algorithm for canonicalizing OpenKG. Which uses the information from the source documents and the embedding of the knowledge graph. We then discussed the algorithm for knowledge graph embedding specific to the openKG. There are some challenges (phrases with same semantic meaning have different nodes in the graph) in OpenKG, that is why we needed different algorithm for openKG. CaRe uses the CESI algorithm along with any knowledge graph embedding algorithm. The main components of the CaRe algorithms are phrase encoder network (for encoding relation phrases) and canonical cluster encoder network (for encoding noun phrases).

In the fourth chapter we discussed one of the important application of the knowledge graph embeddings (Question Answering). We discussed a model that can answer the simple questions using the knowledge graph. We discussed the architecture, important components of the model. Predicate Learning Model which uses Bi-LSTM architecture with attention, which predicts the predicate/relation. We also discussed Head Entity Detection model which reduces the search space by assuming that head entity is the part

of the question. Joint Distance Metric is used to select the fact which is closest to the predicted fact to get the answer to the question.

Apart from this model we discussed challenges in automatic evaluation of the question answering system. We discussed a transformer based approach to evaluate the question answering system. We discussed three configuration of the AVA model. We also discussed which configuration is best out of all three.

Question Answering is the task in which user asks a question and system gives the answer to the question. There are two types of question answering task, close domain (Reading Comprehension) and open-domain question answering. In this chapter we discussed about reading comprehension task. We discussed some common datasets used for training the QA systems. SQuAD is the most popular dataset. Answer to the question in the SQuAD is in the span of the paragraph (which is one of the limitation of SQuAD model). Next we discussed various others datasets, MultiRC which contains questions that requires multiple sentence to answer the question. It contains multiple choice questions. Natural Questions (NQ) is a dataset for close-domain question answering systems, it contains questions, wikipedia links, short and long answers. After discussing some common datasets, we discussed BiDAF model which uses the LSTM with attention, it involves encoding step (which uses character embedding, word embedding and phrase embedding layer), Context to Query attention, Query to Context attention, Modeling and output layer. Next we discussed BERT for reading comprehension. We fine tune BERT language model with additional two output layers, one for predicting start token, other for predicting end token. We then discusses Span BERT which uses different pre-training objective then BERT. Instead of masking 15% of the token. Span BERT masks span of the sentence. Span BERT performs better than BERT in the Question Answering task. We then compared BiDAF and BERT model. Which have many differences but fundamentally they are not so different, BERT uses self attention between context and query, query and context, context and context, query and query, while BiDAF uses context to query attention, and query to context attention.

In the sixth chapter, We first discussed the problem statement, which is to build the question answering system using the knowledge graph. Next we discussed the pipeline of the project. The main components in the pipline are the knowlege graph, Question Classifier and, Pre-trained QA systems. Question Classifier used in the project is BERT based sequence classifier. Pre-trained QA systems are used based on the result of the Question Classifier. BERT is used for answering factoid based questions. T5 is used for answering the non-factoid questions. A brief information is given about the natural language generation module which is rule based. If the answers are long (>40 words), then the summarization module is used to summarize the answer. UI for debugging/Testing is also described, which is made using the streamlit library.

7.2 Conclusion

Knowledge graphs are great resources, it contains the knowledge that machine learning algorithm can use, we still need more powerful algorithms to use the knowledge in the knowledge graph to full extent. Knowledge graph embeddings are a great way to do that. We cannot give words to the machine learning algorithm, we can only give numbers as an input to them. Embeddings are the vector representation of things. Knowledge graph

embeddings algorithms tries to capture whole knowledge graph into vectors representation, therefore investing more time on creating good knowledge graph embedding algorithm can help almost all NLP applications.

7.3 Future Work

Deep Learning algorithms are data hungry. They need lots of data, gathering data needs resources. Algorithms have to learn from lots of data which also takes both time and resources. One of the reason that deep learning needs so much data is they do not have any knowledge integrated with them. They have to learn everything from the data. Integrating knowledge with deep learning methods may reduce the need for the data, and also the performance of the models will also increase. My future work is to learn more techniques through which we can integrate knowledge to deep learning systems.

Bibliography

[1] Limdiwala, Dhaval, Amit Patil, and Pushpak Bhattacharyya. "Survey on Development of Domain-Specific Knowledge Graph.".2019

[2] Reddy, A. Chandra Obula, and K. Madhavi. "A Survey on Types of Question Answering System." IOSR-JCE 19, no. 6.2017.

[3] Miller, George A. "WordNet: a lexical database for English." Communications of the ACM 38.1995

[4] Navigli, Roberto, and Simone Paolo Ponzetto. "BabelNet: Building a very large multilingual semantic network." In Proceedings of the 48th annual meeting of the association for computational linguistics, pp. 216-225. 2010.

[5] Singh, Push, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. "Open mind common sense: Knowledge acquisition from the general public." In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", pp. 1223-1237. Springer, Berlin, Heidelberg, 2002.

[6] https://yago-knowledge.org/getting-started

[7] Wang, Zhen, Jianwen Zhang, Jianlin Feng, and Zheng Chen. "Knowledge graph embedding by translating on hyperplanes." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 28, no. 1. 2014.

[8] Dai, Yuanfei, Shiping Wang, Neal N. Xiong, and Wenzhong Guo. "A survey on knowledge graph embedding: Approaches, applications and benchmarks." Electronics 9, no. 5.2020.

[9] Bordes A, Usunier N, Garcia-Duran A, Weston J, Yakhnenko O. Translating embeddings for modeling multi-relational data. InNeural Information Processing Systems (NIPS). 2013 .

[10] Lin, Yankai, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. "Learning entity and relation embeddings for knowledge graph completion." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29, no. 1. 2015.

[11] Nickel, Maximilian, Lorenzo Rosasco, and Tomaso Poggio. "Holographic embeddings of knowledge graphs." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 30. No. 1. 2016.

[12] Nickel, Maximilian, Volker Tresp, and Hans-Peter Kriegel. "A three-way model for collective learning on multi-relational data." Icml. 2011.

[13] Kolda, Tamara G., and Brett W. Bader. "Tensor decompositions and applications." SIAM review 51, no. 3. 2009.

[14] Socher, Richard, Danqi Chen, Christopher D. Manning, and Andrew Ng. "Reasoning with neural tensor networks for knowledge base completion." In Advances in neural information processing systems, pp. 926-934. 2013.

[15] Dettmers, Tim, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. "Convolutional 2d knowledge graph embeddings." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1. 2018.

[16] Nguyen, Dai Quoc, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. "A novel embedding model for knowledge base completion based on convolutional neural network." arXiv preprint arXiv:1712.02121 2017.

[17] Vashishth, Shikhar, Prince Jain, and Partha Talukdar. "Cesi: Canonicalizing open knowledge bases using embeddings and side information." In Proceedings of the 2018 World Wide Web Conference, pp. 1317-1327. 2018.

[18] Gupta, Swapnil, Sreyash Kenkre, and Partha Talukdar. "Care: Open knowledge graph embeddings." In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 378-388. 2019.

[19] Huang, Xiao, Jingyuan Zhang, Dingcheng Li, and Ping Li. "Knowledge graph embedding based question answering." In Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pp. 105-113. 2019.

[20] Chen, Anthony, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. "Evaluating question answering evaluation." In Proceedings of the 2nd Workshop on Machine Reading for Question Answering, pp. 119-124. 2019.

[21] Vu, Thuy, and Alessandro Moschitti. "AVA: an Automatic eValuation Approach to Question Answering Systems." arXiv preprint arXiv:2005.00705. 2020.

[22] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805. 2018.

[23] http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture11-qa-v2.pdf

[24] Khashabi, Daniel, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. "Looking beyond the surface: A challenge set for reading comprehension over multiple sentences." In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pp. 252-262. 2018.