Efficient Algorithm for Answering Fact-based Queries Using Relational Data Enriched by Context-Based Embeddings

by

A. Aziz Altowayan

December 12, 2019

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

at

Seidenberg School of Computer Science and Information Systems

PACE UNIVERSITY
NEW YORK, NY
We hereby certify that this dissertation, submitted by A. Aziz Altowayan, satisfies the dissertation requirements for the degree of Doctor of Philosophy in Computer Science and has been approved.

<table>
<thead>
<tr>
<th>Name</th>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Professor Lixin Tao</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dissertation Committee Chair (Advisor)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Professor Li-Chiou Chen</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dissertation Committee</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Professor Anthony Joseph</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dissertation Committee</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Seidenberg School of Computer Science and Information Systems
Pace University, December 12, 2019
Abstract

Intelligent conversational systems - such as question answering and chatbots - are becoming a more critical component of today’s AI in areas ranging from health, medicine, and security, to personal assistants, and other domains.

In one way or another, the core of the design in such systems relies heavily on figuring out the relationships between the different components of the specific domain. For example, knowing that Tom Cruise has played in a movie called Top Gun would mean that he is, also, one of the cast in that movie, and so on. This information, for instance, constitutes that there is a relation between Tom Cruise and Top Gun movie.

Today, this relational knowledge is harvested and available in structured data form known as Knowledge Graphs - collections of connected facts about the world. Knowledge graphs comprise billions of facts collected by web-scale knowledge extraction projects such as NELL, Free-Base, Google Vault, WikiData, and others. One problem is how to enable computers to, efficiently, use such vast knowledge banks.

For intelligent systems to make use of this knowledge, knowledge graphs need to be modeled (or represented) in a machine-comprehensible, meaningful, and processable way. Knowledge Graph Embeddings “KGE” is a method for delivering such a meaningful representation for relational knowledge. KGE is partially inspired by Neural Word Embeddings - a recent widely successful approach for language modeling in Natural Language Processing NLP.

However, existing KGE methods do not fully-leverage the core ideas behind word embeddings. The Distributional Hypothesis is one of the main reasons word embeddings work. This hypothesis states that words with similar meanings occur in similar contexts. Such an essential notion is not accounted for “unconsidered” in the current knowledge graph embedding approaches.

In this dissertation, we introduce a Context-Based Knowledge Graph Embeddings. An approach for modeling knowledge graphs by leveraging the semantic similarities of relationships between knowledge graphs triplets. In which we borrow the idea of “similar words happen in similar contexts,” and apply it to the knowledge graphs triplets as “similar relations happen in similar contexts.” Our approach is a hybrid of an existing embedding method and the hypothesis mentioned above.

Further, we present our published experiments for learning and building embedding models. Our validations show that a simple embedding model is capable of beating state-of-the-art approaches that rely on hand-crafted features; in terms of capturing the linguistic similarities and meanings.

Finally, we introduce our factoid-question answering algorithm that leverages the embeddings of knowledge graphs for answering simple fact-based questions. NLP techniques, such as Named Entity Recognition and Relation Extraction, are applied in which we propose a new approach suited for knowledge base data structure. We built a web application to demonstrate the efficiency of this algorithm. The app works with any customized dataset. It is open-sourced and made available for general use.
Keywords — Relational Knowledge Representation, Ontologies, Knowledge Bases, Knowledge Graphs Embeddings, Machine Learning, Natural Language Processing, Question Answering
Acknowledgements

Alhamdulillah.

Thanks to my advisor Dr. Lixin Tao and to the committee members for their help and time. I would also like to thank all the professors, classmates, and people whom I met during this journey for their assistance and inspiration. Also, I feel lucky for living and studying in New York City amid such a rich and vibrant tech community. Hence, a special thanks to the Python and Machine Learning communities in NYC for whom I owe much of my professional knowledge and experience.

Finally, I’d like to extend my gratitude to my financial sponsor, the Saudi Arabian Cultural Mission (SACM) in Washington, D.C.
To my mother
Contents

List of Tables viii
List of Figures ix

1 Introduction 2
   1.1 Motivation ........................................ 5
   1.2 Use Case ........................................ 6
   1.3 Problem Statement ................................. 8
       1.3.1 Dissertation .................................. 9
       1.3.2 Contributions ................................ 10

2 Background and Related Work 11
   2.1 Distributional Semantics and Distributed Representations .................. 11
   2.2 Language Modeling and Relational Representation .......................... 14
   2.3 Relational Embedding ................................ 21
       2.3.1 Embedding for relation representation .................... 22
       2.3.2 Embedding for relation classification ...................... 24
   2.4 Ontology (concepts) and Knowledge Graphs (entities) ...................... 26

3 Representing Ontology Relationships 27
   3.1 Supporting Part-Whole Relations with OWL and Description Logics (DLs) .... 27
   3.2 Supporting Multi-Type Relations with Machine Learning .................... 42
       3.2.1 Employing embedding methods to represent Relationships and Entities: .. 42
       3.2.2 Visualizing entities and their relationships ...................... 43
       3.2.3 Experimental observations ................................ 44

4 Embedding-Based Representation: Learning Embeddings for Relational Data 47
   4.1 Building Embeddings From Raw Text ................................ 47
       4.1.1 Embeddings to replace hand-crafted feature extraction in text ......... 47
       4.1.2 Experiment design, implementation, and validation .................... 49
   4.2 Context-Based Approach For Learning Knowledge Graph Embeddings ......... 59
       4.2.1 Data sources and structure ................................ 59
       4.2.2 Data collection and description ................................ 60
       4.2.3 Hybrid context-based training algorithm for relational data ............ 61
       4.2.4 Validation process ...................................... 64

5 Experimental Evaluation and Validation 65
   5.1 Evaluating the Quality of the Embedding Vectors .......................... 65
   5.2 Applying KGE for Answering Fact-based Questions .......................... 77
   5.3 KGE QA System ........................................ 77
       5.3.1 Algorithm and Application Screenshot ............................ 77
       5.3.2 How we build ENT and REL models ............................ 81
       5.3.3 System description and design ................................ 83
       5.3.4 System performance and accuracy ................................ 88
       5.3.5 KGE QA dataset description ................................ 88
   5.4 System setup and project configurations .................................. 98

6 Conclusion 99

7 Bibliography 100
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One can use intuition to easily guess some possible word(s) that can best complete the sentence based on its context. Likewise, NLM exploit contexts similarity “Distributional Hypothesis” to predict the correct word for any given training example.</td>
</tr>
<tr>
<td>2</td>
<td>Accuracy of current state of the art on FB15k dataset (Raw).</td>
</tr>
<tr>
<td>3</td>
<td>Statistics of the benchmark dataset for relation classification.</td>
</tr>
<tr>
<td>4</td>
<td>Results of embedding-based relation classification. The symbol means that one of the classes \textit{Other} (which introduces noise to the dataset) was removed.</td>
</tr>
<tr>
<td>5</td>
<td>Ontology vs. Knowledge Graph</td>
</tr>
<tr>
<td>6</td>
<td>Corpus collections and sources</td>
</tr>
<tr>
<td>7</td>
<td>Our collection of Twitter datasets and the source of each.</td>
</tr>
<tr>
<td>8</td>
<td>Methods performance compared on the MPQA subjectivity of Standard Arabic.</td>
</tr>
<tr>
<td>9</td>
<td>Classifiers and their scores on each dataset. (Note: Rec., Prec., and MAcc. scores are the average of both positive and negative classes).</td>
</tr>
<tr>
<td>10</td>
<td>Statistics of the experimental datasets we use.</td>
</tr>
<tr>
<td>11</td>
<td>Relation type frequencies in WN18</td>
</tr>
<tr>
<td>12</td>
<td>Sample KB triplets for “molecule” entity in WN18</td>
</tr>
<tr>
<td>13</td>
<td>An example training dataset.</td>
</tr>
<tr>
<td>14</td>
<td>Statistics of the training text (corpus).</td>
</tr>
<tr>
<td>15</td>
<td>Embedding vectors compared.</td>
</tr>
<tr>
<td>16</td>
<td>Vocabulary Size, Average Retrieval Errors, and Classifiers Performance with each model.</td>
</tr>
<tr>
<td>17</td>
<td>Summary on the final results for embedding models’ accuracy and classification performance</td>
</tr>
<tr>
<td>18</td>
<td>An example of a raw FB15K triplets</td>
</tr>
<tr>
<td>19</td>
<td>An example of FB15K triplet with meaning descriptions</td>
</tr>
<tr>
<td>20</td>
<td>stats of the filtered FB15K datasets</td>
</tr>
<tr>
<td>21</td>
<td>stats of the filtered relationships in FB15K</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sample knowledge graph. Nodes represent entities, edge labels represent types of relations, edges represent existing relationships [83].</td>
</tr>
<tr>
<td>2</td>
<td>Machine learning: a programming paradigm that equips machines with the capability of self-learning.</td>
</tr>
<tr>
<td>3</td>
<td>An example fragment of a KB.</td>
</tr>
<tr>
<td>4</td>
<td>In text representations, state of the art for modeling the semantics of words is a combination of NLM and DH (at the top). On the other hand, current state-of-the-art for modeling the semantics of relations does not incorporate DH. Inspired by NLM for word representations, our proposed approach to relational representations extends current NLMs to account for distributional hypothesis.</td>
</tr>
<tr>
<td>5</td>
<td>The surrounding words of the word <em>banking</em> can be used to infer its meaning. Distributional similarity is regarded as one of the most successful ideas in modern statistical NLP.</td>
</tr>
<tr>
<td>6</td>
<td>Hypothetical example to demonstrate how “dense-vector” distributed representations can share attribute among different concepts, in compare to the “one-hot” sparse representations.</td>
</tr>
<tr>
<td>7</td>
<td>Skip-gram representations of training samples (at each training iteration) with a window size = 1.</td>
</tr>
<tr>
<td>8</td>
<td>TransE represents a triplet relation r as a translation between head h and tail t in the vector space. (image credit [104]).</td>
</tr>
<tr>
<td>9</td>
<td>Keet’s Taxonomy of basic mereological and meronymic part-of relations. Dashed lines indicate that the subtype has additional constraints on the participation of the entity types; ellipses indicate several possible finer-grained extensions to the basic part-whole relations.</td>
</tr>
<tr>
<td>10</td>
<td>Simple Scenario For PartOf and IS-A relations.</td>
</tr>
<tr>
<td>14</td>
<td>We use HermiT 1.3.8 reasoner (built-in protege) to evaluate the representation reasoning of both (a) our approach (b) the current approach for ontology example 3. And the results (as anticipated) are the same.</td>
</tr>
<tr>
<td>15</td>
<td>Visualization of The Simplified Ontology in Listing 7: Using Our Module We Visualize The Inferred Model Directly From The Simplified Representation.</td>
</tr>
<tr>
<td>16</td>
<td>An example fragment of WordNet KB in a graph generated by our PyGraph wrapper.</td>
</tr>
<tr>
<td>17</td>
<td>Ranking loss with different embedding size. From top to bottom embedding size ( k = {100, 50, 20} )</td>
</tr>
<tr>
<td>18</td>
<td>Slight improve in ranking loss after increasing the batch size from 100 (orange) to 400 (blue).</td>
</tr>
<tr>
<td>19</td>
<td>Visualizing relation embeddings in a 3D vector space model (dimensionality reduction is based on t-SNE). In the sample: appears top 5 nearest neighbors (using cosine-similarity) of the relation <em>part_of</em>.</td>
</tr>
<tr>
<td>20</td>
<td>Learning curves of NaïveBayes and SupportVectors on Sentiment and Subjectivity datasets.</td>
</tr>
<tr>
<td>21</td>
<td>Classifiers ROC on each dataset.</td>
</tr>
<tr>
<td>22</td>
<td>Letters frequencies as they appear in text8 and IMDB.</td>
</tr>
<tr>
<td>23</td>
<td>Embeddings results on the word analogy task (out of the total 19544 questions), fig. a. is the number of questions seen and fig. b. is the percentage of the questions seen.</td>
</tr>
<tr>
<td>24</td>
<td>Results on the topics accuracy from word analogy task.</td>
</tr>
<tr>
<td>25</td>
<td>The overall accuracy on each embedding model on all the 14 topics in the analogy test.</td>
</tr>
<tr>
<td>26</td>
<td>Sentiment classifiers score with each embeddings. a) embedding models wise results, and b) classifiers wise results.</td>
</tr>
<tr>
<td>27</td>
<td>The main user interface of the KGE QA system.</td>
</tr>
</tbody>
</table>
Screenshot of the usage information ........................................ 79
Example1: uncleaned FB15K dataset: example question and answer ........................................ 79
Example2: sample question and answer ........................................ 80
Example2: visualizing the closest relations based on the input question ........................................ 80
Example2: visualizing the closest entities based on the input question ........................................ 81
Workflow for building ENT.vec and REL.vec models with an example ........................................ 82
Example for building new KGE models from the UI ........................................ 82
Created new embedding models for entities and relations ........................................ 83
Example of CLI interface for the QA system. Same question can be asked in different ways ........................................ 84
Conceptual design of the factoid QA system ........................................ 85
KGE QA workflow for answering a question ........................................ 89
Example query with its relevant part in the Knowledge Graph ........................................ 90
Conceptual approach for finding answers ........................................ 90
Computer ontology graph ........................................ 93
Film ontology graph ........................................ 93
People ontology graph ........................................ 94
Location ontology graph ........................................ 94
Complete ontology graph of our QA system ........................................ 95
Visualizing the closest 20 relations in our REL.vec to the word “wrote” ........................................ 97
localhost:8501 to get started in the browser ........................................ 98
NOTE: Despite the dissertation title, in our work, we put more emphasis and efforts (i.e. our contribution is mostly) in the embedding part; which tackles the crux of the relational knowledge representation problem.
1 Introduction

I am convinced that the crux of the problem of learning is recognizing relationships and being able to use them.

Christopher Strachey, in a letter to Alan Turin in 1954 [32].

In traditional search systems, e.g. search engines, search results are derived based on what the searched term is, not what it means. Reliance on such symbolic representation is not future promising. Hence, modern search engines have been shifting search mechanism from the conventional textual and rule-based approaches to more semantic and context-aware approaches. This shift can transform search from a static information engine to a dynamic knowledge engine. Knowledge Graphs (KGs), i.e. graph structured knowledge bases (KBs), are at the heart of such modern approach. Applications for KGs are utilized for improving the capability of knowledge representations in knowledge inference, fusion, and completion. For instance, in KGs, entities (or objects\(^1\)) are connected to each other via relationships. By understanding the relationships between things—be it between players and sports or diseases and symptoms—intelligent systems can do a better job of understanding what it is exactly one is searching for [94].

Knowledge bases store factual information about the real-world in form of binary relations between entities\(^2\). The purpose of KBs is to convey some commonsense knowledge about either everyday life or expert knowledge about an application area to an artificial intelligence system [43]. This form of knowledge representation plays a crucial role in many intelligent systems and search-based areas such as search engines and medical-diagnosis applications. For example, improving search results with semantic information from knowledge bases is an important step for transforming text-based search engines into semantically aware question answering services [83]. Modeling the relationships between entities allow computers to acquire new knowledge from existing examples, deduce conclusions based on existing facts, formalize facts about objects, and/or understand how objects interact with each other.

Many knowledge bases have been created in the past years, including Wordnet [72], GeneOntology [31], DBpedia [14], YAGO [97], Freebase [22], NELL [28], Google Knowledge Graph [94], and most recently Knowledge Vault [33]. Through time, KBs have accumulated and stored a large number of facts about the world. These multi-relational data form directed graphs (of knowledge) whose nodes

\(^1\)An object could be anything, e.g. a person, a book ... etc.

\(^2\)Relations in KBs are known as triplets.
correspond to entities and edges correspond to relations between entities; see figure 1. Such graphs play a pivotal role in many areas [24]. For example, they are used to enhance search engines results (Google’s Knowledge Graph and Microsoft’s Satori). Decision support systems in healthcare, such as LinkedLifeData [74], is another prominent example of the value of knowledge graphs.

![Sample knowledge graph](image.png)

Figure 1: Sample knowledge graph. Nodes represent entities, edge labels represent types of relations, edges represent existing relationships [83].

However, as KBs grow it becomes hard to manipulate and handle these structured graphs. The dimensionality of one knowledge base can grow to as large as $10^8$ entities and $10^6$ relation types (e.g. Freebase contains 40M entities, 35K relations, 637M facts). Further, noisy data can be introduced by wrong relations and entities. Also KBs are far from been complete, with many incomplete facts or few valid links (e.g. in Freebase nationality for 71% of persons missing). This incompleteness has stimulated research into predicting missing links, a prominent problem in statistical relational learning known as “link prediction”. Tackling these issues is a key to automatically understand the structure of large knowledge bases.

"Early AI systems sought to hard-code knowledge about the world in formal languages."

In [43], they explain how early artificial intelligence projects employ human experts to manually encode knowledge into formal languages. A logic-reasoner program can reason about these formal statements using inference rules. This is known as the rule-based approach to AI. In such systems, people struggle to devise formal rules to accurately describe the world with minimum complexity. As a result of this approach’s limitations (e.g. scalability, adaptivity, handling complexity), none of these systems has led to a major success. Cyc$^3$ project is one of the most famous examples of such approach.

The difficulties resulted from hard-coding knowledge suggest that an intelligent system needs to be

---


4Image credit: [29]
capable of self-learning and be able to extract knowledge (and rules) from raw-data. This capability is known as Machine Learning. A machine learning system is trained rather than explicitly programmed. Presented with many examples relevant to a certain task, it discovers statistical structure in these examples which eventually allows the system to come up with rules pertaining that task [29], see figure 2. And as such system would rely on a learning algorithm for getting trained; it turns out that the performance of the learning algorithm depends heavily on the representation of the raw data. In the context of this work, knowledge base triplets are the raw data.

**Statistical semantics.** In 1957, the English linguist John Firth proposed his now-popular notion about the context-dependent nature of meaning [37]. His hypothesis, which later became the basis of an entire research area called *Distributional Semantics*, suggests that words that are used and occur in the same contexts tend to have similar meaning. In other words, similar words usually fall under similar contexts. In linguistics, this is known as “*Distributional Hypothesis*”. This underlying idea is at the heart of the recent *neural word embedding* models.

**Word embeddings.** Recently, deep learning\(^5\) have been very successful in tasks that require understanding of natural language (i.e. in Language Modeling, Machine Translation, and Natural Language Processing). The very reason behind that success is the use of neural *embeddings* (also called distributed representations of words) to represent the semantic knowledge of individual words and concepts. Neural embeddings refer to a class of word embedding methods that are based on neural networks. Although such models have been around for more than a decade ago [19], they had not become a major success until recently when they were efficiently designed (along with the optimized training process) based on the assumption of distributional hypothesis [68]. Due to this success in distributed representations of words, along with the followup work [71, 58, 84, 21, 52], neural language modeling has emerged as the main spectrum for *distributional semantic* models.

\(^{5}\)Deep learning is a special (subset) class of machine learning.
1.1 Motivation

Relation semantics. A natural next step, beyond representing the semantics of individual words, is to model and represent the relations between these words and understand how words “semantically” relate to one another. Thus, a current research frontier is to develop embeddings for relations between words and facts. Search engines are already exploring this area; however much more remains to be done to improve these advanced representations [43]. One example on how to make use of these representations is to predict which triplets, in a knowledge base, are likely to be true [24, 82].

1.1 Motivation

"The connection is indispensable to the expression of thought. Without the connection, we would not be able to express any continuous thought, and we could only list a succession of images and ideas isolated from each other and without any link between them.”

One way people understand a written text is by identify the semantic relations which connects the entities described in that text. Likewise, a system which aspires to human-like performance need to have the capability to identify, and learn from, the semantic relations in the texts it processes. Understanding even a simple sentence such as “Opportunity and Curiosity find similar rocks on Mars” requires recognizing relations (rocks are located on Mars, singled by the word on) and drawing on already known relations (Opportunity and Curiosity are instances of the class of Mars rovers) [77].

Recently, the natural language understanding community has shown a renewed interest in deeper semantic analyses, among them automatic recognition of semantic relations between pairs of words. Automatic recognition of semantic relations is an important task with many potential applications. Some of these applications include but not limited to Question Answering, Semantic Network Construction, Language Modeling, Machine Translation, Information Retrieval, Information Extraction, Text Summarization, Paraphrasing, and Recognizing Textual Entailment.

An interesting research direction is determining how distributed representations can be trained to capture the relations between entities. One way to make use of these advanced distributed representations is to use them as features in semantic relation classification.
1.2 Use Case

Web-scale Knowledge Bases (KBs) provide a structure representation of the world knowledge. However, as we discussed in the introduction, KBs are incomplete, noisy, and contain many missing entries. This has stimulated research into finding new approaches to tackle such issues. Link prediction task, a main problem in Statistical Relational Learning [39], is one way to address KBs problems. The task of link prediction is regarded as a key for understanding the structure of large knowledge bases [101].

**Link Prediction Task.** Link prediction refers to the task of predicting the existence of typed edges in the graph (i.e. triplets). To some extent, link prediction is also considered as a standard metric for evaluating the quality of KB representation models. In the context of KBs, link prediction is known as knowledge base completion. For instance, consider the example KB fragment in figure 3 which contains an unknown (undefined) triplet. A KB representation model can use related facts to predict (infer) incomplete or missing facts.

To put this into more context, let’s consider the following scenario about an incorrect fact. Let’s assume an information extraction system returns an incorrect fact claiming that Lionel Messi’s profession is actor. And just for the purpose of this illustration, let’s also assume that the true profession of Lionel Messi was not already stored in the knowledge base. A KB representation model can use related facts about Messi, (such as the fact that he won ballon d’Or prize multiple times) to infer that the new fact is unlikely to be true and should be discarded\(^6\). This also allows us to automatically grow KB.

![Figure 3: An example fragment of a KB.](example.png)

\(^6\)a modified example from [83]
1.2 Use Case 7

Link prediction using relation embeddings. Current state of the art approaches for KB completion are embedding-based models. In contrary to other models such as markov-based approaches, embedding models generalize to large KBs. To do that such, entities and relation are embedded in a dense ... There have been multiple work in this area. Since a comprehensive literature review is beyond the proposal’s scope, we focus only on work with most significant results, such work include [80, 23, 24, 109, 82, 62, 79, 78, 38]. The quality of these methods’ embeddings is evaluated based on their performance on link prediction task. In section 2.3, we discuss these methods and their performance in greater details.

The recent success of word embeddings is attributed to two key components. First, the use of neural language models NLMs (see section 2.2). Second, the idea of distributional semantic hypothesis (see more about it in section 2.1) which is the underlying assumption for training word vectors using NLMs.

What is the issue with current approaches? While current relation embeddings models are already trained using NLMs, they do not actually assume distributional similarities over the relation types of KBs triplets. Such lack of an underlying assumption for a semantic theory is a major theme in current work. Therefore, we think that the shortcoming of current models is due to the lack of a semantic theory underlying their training process; which in turn resulted in low accuracy and performance.

Can we do better? Although there have been some progress made due to the fact of using NLMs, the accuracy of current approaches is still far from being useful for real world applications, see tables 2, and 4. Hence, more work needed in order to improve these advanced representations.

Why distributional similarity is absent from the architecture of current relation embedding models? In the case of word representations, it is obvious how distributional hypothesis fits given the succession of words in a sentence or a corpus. However, in the case of KB relation representations, it is not apparent how the consecutiveness of words can be applied given the discrete and independent nature of knowledge base triplets.

To tackle this problem, we propose to incorporate the distributional hypothesis assumption into the training process, see figure 4. In order to do that, we treat triplets of knowledge bases just like words in a corpus. Such that the relation type of a triplet is the link that connect triplets as a sequence. To better understand this idea, refer to the detailed approach description in section 4.2.3.

---

7 corpus is a large collection of written texts.
1.3 Problem Statement

Neural Word Embeddings is a breakthrough in natural language processing. It has been proven as the - currently - most efficient technique for capturing the semantic and syntactic similarities between linguistic items. Applications such as web search systems rely massively on the semantic interpretations of web content. In numerous situations, relational models have been proven to improve the results of semantic search when relational information is available. While the accuracy of those models is pivotal for subsequent tasks in semantic search engines, the performance of existing relational learning models is still far from being useful to leverage in real-world applications.

We argue that the absence of a semantic theory in current relation embedding models has resulted in low accuracy and, thus, incompetent performance of these models. As a result, current approaches are unable to address representation issues in knowledge bases such as large dimensionality and incompleteness.

Inspired by the success of neural word embeddings as an example, we believe that the way to improve current relation embeddings is by integrating context similarity in the training process. Thus, the goal in this dissertation work is to take full advantage of the distributional similarity theory in order to improve the accuracy of relational knowledge representation models.

Figure 4: In text representations, state of the art for modeling the semantics of words is a combination of NLM and DH (at the top). On the other hand, current state-of-the-art for modeling the semantics of relations does not incorporate DH. Inspired by NLM for word representations, our proposed approach to relational representations extends current NLMs to account for distributional hypothesis.
1.3 Problem Statement

1.3.1 Dissertation

*Distributional Hypothesis* is one of the core ideas/assumptions behind the recent Neural Word Embeddings. Its objective is to quantify and capture the semantic similarities between words. In this dissertation, we borrow and apply the same hypothesis to build a representation model for Relational Knowledge data such as Knowledge Graphs. The goal is to build a relational embedding model capable of capturing and representing the semantic similarities between relationships.

Below are the topics and methodology discussed and utilized in this dissertation (in the same work order):

- Knowledge Representation
- Relational Data: Web Ontologies and Knowledge Base
- Semantic Web Tools and Technology
- Knowledge Graphs and Representation Learning
- Machine Learning, Natural Language Processing, and Neural Networks
- Distributional Semantic Theory and Neural Word Embeddings
- Question Answering Systems
  - Named Entity Recognition “NER” and Entity Linking
  - Relation Extraction

The dissertation statement can be summarized in the following paragraph:

Contributed to *neural* language models, the recent advancement in language modeling has led to several successes in understanding natural language. Mainly based on embedding methods, the architecture design of neural language models is based on the assumption of the distributional semantics theory, also known as the “Distributional Hypothesis”. This notion is absent from the current Knowledge Graphs Embeddings KGEs. We, thus, believe that assuming distributional similarity between the Knowledge Graph relations “triplets” can improve the current KGE methods.

Unlike existing work, the proposed approach leverages the principle of distributional hypothesis as a solution for how distributed representations can be trained to capture *relations* between entities. Figure 4 illustrates the distinction between the proposed approach and existing work.
1.3.2 Contributions

The dissertation work and contributions in a nutshell include:

- Developed a Context-Based Embedding algorithm for training Knowledge Graphs to represent entities and their relationships as Vector Space Model “VSM”
- Designed and implemented intrinsic and extrinsic evaluation methods to estimate the quality of the embeddings
- Designed and implemented a novel framework for Fact-based question answering that leverage the embeddings of knowledge graphs (and demonstrate how to use them in a real-world application)
2 Background and Related Work

This chapter introduces the knowledge on which the dissertation work is based on. It is structured into two parts. The first one (sections 2.1 and 2.2) introduces the general background and methodology behind the proposed work. In the second part (section 2.3) we discuss work and techniques that are directly related to the problem of learning semantic relations.

When it comes to representing natural languages, the natural language processing community has a tradition of thinking in terms of *distributional semantics*, while on the other hand, the neural networks community has a tradition of thinking in terms of *distributed representations* [41]. In first part of section 2.1 we explore the former, and in the second part of the same section we delve in the latter approach. While, in section 2.2, we discuss *neural language models* where the two worlds meet.

2.1 Distributional Semantics and Distributed Representations

**Distributional semantics hypothesis**

A human brain can easily identify and differentiate between different objects. It is even capable of drawing a connection between two objects based on how they relate to each other. The way these connections are formed is governed by the nature (meaning) of the target objects. For example, in the simple sentence “The car’s engine is broken.” the brain can easily recognize the relationship that “engine” is part of “car”. With many more related objects, the brain forms sort of relational maps or networks of connected objects. The brain relies on these interconnected networks to reason and draw conclusions about various decisions (problems). This phenomena mental capability of the human brain is the result of accumulated knowledge learned throughout the human’s lifetime.

When reading a written text, the brain employs its magical ability to establish relationships between words. Again, the meaning of a word is the main criteria for determining how a word connects (relates) to other words. For example, these meanings (or semantics) are derived naturally via a cognitive process happen in the brain\(^8\). This very particular cognitive ability is what we hope computers to achieve.

*Distributional semantics* provide a way for quantifying the meaning of words. Thus, opening doors

---

\(^8\)we do not actually understand much about how this process happen on an algorithmic level; computational neuroscience is the primary field concerned with this endeavor.
for a computer to capture and understand words meanings and, then, utilize this understanding to establish connections (relations) between words on its own.

**Distributional Hypothesis**

"You shall know a word by the company it keeps."

Firth, J. R. 1957

From a higher perspective, *Distributional Hypothesis* is a notion about the nature of language and meaning. It was first hypothesized by [37] with a famous quote that states “You shall know a word by the company it keeps”, figure 5. Often, it is expressed in terms such as “words that appear in the same contexts share semantic meaning”; “words which are similar in meaning occur in similar contexts” [87]. For instance, often when people encounter a sentence that has an unknown word, they intuitively infer the meaning of the word based on the context in which it occurs. The word *wampinuk* in *Marco saw a hairy little wampinuk crouching behind a tree* is an example of such cases [41].

The basic idea of *distributional hypothesis* is that there is a correlation between distributional similarity and semantic similarity. Consequently, we can utilize the former to quantify “estimate” the latter. In meaning acquisition, the distributional properties of linguistic entities are treated as the building blocks of semantics [89].

Figure 5: The surrounding words of the word *banking* can be used to infer its meaning. Distributional similarity is regarded as one of the most successful ideas in modern statistical NLP.

This basic idea is very pivotal to the development of semantic representations; which in turns allowed researchers to develop machine-learnable semantic features.

Distributional similarities have been the driving factor behind the development of Vector Space Models (VSMs) in natural language processing. VSMs embed words in a dense-continuous vector space such that similar words are mapped to nearby points in the space. Although computers understanding for the meaning of human language is very limited, the use of VSMs for semantics

---

9Credit: Stanford’s cs224d (Socher, 2016)
10https://www.tensorflow.org/tutorials/word2vec
processing is beginning to address that limitation [102].

There are two different categories of VSMs approaches that leverage the principle of distributional hypothesis, 1) count-based methods, e.g. Latent Semantic Analysis “LSA”; and 2) predictive methods, e.g. neural probabilistic language models “NLMs”. An extensive evaluation about the two categories and the distinction between them is discussed in details in [18]. The approach we follow in this dissertation falls under the second category.

**Distributed representations**

One of the most important tools in *representation learning* is to represent concepts as a composition of many elements that can be set separately i.e. “distributed representation”. Distributed representations strength lies in their capacity to describe (express) many different concepts in a minimal shared space. For example, they can use \( n \) features with \( k \) values to describe \( k^n \) distinct concepts. Neural networks with multiple hidden layers make use of this strategy [43].

To put this idea into context, lets demonstrate it through an example. A vector of \( n \) binary features can take \( 2^n \) different configurations, where each can be mapped to a distinct concept and potentially corresponds to a different region in the input space. In contrast, in one-hot representation (a.k.a “symbolic representation”) there could be only \( n \) possible configurations. Accordingly, symbolic representations would require \( n \) feature detectors for every \( n \) input symbols. The sparsity of such approach (one-hot) can easily suffers from the “curse of dimensionality” when the input symbols are very large.

Distributed representation is superior to symbolic representation in its ability to *generalize* across multiple sets of concept. **Generalization** arises due to shared attributes between different concepts. To better explain this ability, lets look into the following case. Although the words "cat" and "dog" appear different syntactically, they actually share some semantic similarities. So if one can associate them with a meaningful distributed representation, many attributes about the *cat* can be said about *dog*. Whereas in one-hot representations, it is not possible to express these similarities since words are treated as discrete atomic symbols. This difference is better illustrated in 6. For a comprehensive review on the symbolic vs. distributed representations and the key differences between them in the context of NLP, see [36].

As we see in the figure 6, dense vector representations allow us to measure the similarity between

\(^{11}\)one-hot is a binary vector with \( n \) bits that are mutually exclusive (only one active at a time).
### 2.2 Language Modeling and Relational Representation

#### Neural language modeling and word embedding

**What is language modeling?** The goal of a language model is to learn a probability distribution over a sequence of words from a given dictionary $V$. The joint distribution of a sequence of tokens $T$ given their past is defined as the product of their conditional distribution [44]. That is for a sequence of $T$ words $w_1, \ldots, w_T \in V^T$, its probability distribution is given by:

$$ P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t|w_{t-1}, \ldots, w_1) \quad (1) $$

Neural Language Models, on the other hand, (or NLMs) are a class of language models based on neural networks which solve some of the shortcomings of traditional language models. Known as
context-predicting models, NLMs were designed by [19] to reduce the impact of *curse of dimensionality* through exploiting their ability to learn distributed representations.

In predictive models, a word is predicted directly from its neighbors and take the form of learned small, dense embedding vectors\(^{12}\) that are regarded as parameters of the model [1].

Class-based \(n\)-gram models (e.g. bag-of-words, and symbolic representation approaches in general) are common for representing text features. However, they have two major shortcomings. They ignore words’ semantics, and they fail at preserving the order of words [58].

On the other hand, NLMs are able to address both problems. Such that NLMs can recognize the similarity between two words without losing the ability to encode each word as distinct from the other. Further, NLMs share statistical strength between one word (and its context) and other similar words and contexts. This sharing happens as follows. The model learns distributed representation for each word; these representations then allow the model to treat words that have features in common similarly. For instance, when the two words *cat* and *dog* each map to representations that share multiple attributes, the model can then utilize sentences that contain the word *dog* to make predictions in sentences that contain the word *cat*, and vice versa; see table 1. Because many such attributes can exist, there are many ways in which generalization can take place, and transferring information between semantically related training sentences [43].

Table 1: One can use intuition to easily *guess* some possible word(s) that can best complete the sentence based on its context. Likewise, NLM exploit contexts similarity “Distributional Hypothesis” to *predict* the correct word for any given training example.

<table>
<thead>
<tr>
<th>Incomplete sentence</th>
<th>Possible words</th>
</tr>
</thead>
<tbody>
<tr>
<td>She traveled by ____.</td>
<td>car, bus, plane</td>
</tr>
<tr>
<td>The capital of ____ is ____</td>
<td>(Japan, Tokyo), (Germany, Berlin)</td>
</tr>
<tr>
<td>The _____ has four legs.</td>
<td>dog, horse, cat, elephant</td>
</tr>
</tbody>
</table>

**Neural Model.** Generally, a neural language model is exploited to obtain the joint probability of sequences of words. Where the probability function is expressed in terms of the product of conditional probabilities of the next word given the previous words [19]. This function has multiple parameters that can be tuned to maximize the log-likelihood of the training data. Consider the training set as a sequence \(w_1 \ldots w_T\) of words such that \(w_t \in V\), and the vocabulary \(V\) is a large but finite set. In [19], the objective of such model is to learn \(f(w_1,\ldots,w_{t-n+1}) = \hat{P}(w_t|w_1^{t-1})\). Where they use softmax in the output layer to obtain positive probabilities that sum up to 1 i.e.

\(^{12}\)It worth mentioning that using distributed representations in NLP is not, however, restricted to neural networks only. Graphical models also have distributed representations in the form of multiple latent variables [73].
\[ P(w_t|w_{t-1}, \ldots, w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_{i=1}^{n} e^{y_i}}. \]

A general form for this model is described in [43] as follows. Suppose \( h \) is the final hidden layer which is used to predict the output probability \( \hat{y} \). The transformation from \( h \) to \( \hat{y} \) is parameterized with weights \( W \) and biases \( b \) which are learned by the model.

\[ a_i = b_i + \sum_j W_{ij} h_j \quad \forall i \in \{1, \ldots, |V|\} \quad (2) \]

The **softmax function** is then used as a standard map from \( \mathbb{R}^V \) to a probability distribution:

\[ \hat{y}_i = \frac{e^{a_i}}{\sum_{j=1}^{|V|} e^{a_j}} \quad (3) \]

Such that

\[
\begin{cases} 
  e^{a_i} & \text{Exponentiate to make positive} \\
  \sum_{j=1}^{|V|} e^{a_j} & \text{Normalize the vector} \left( \sum_{k=1}^{n} \hat{y}_k = 1 \right) \text{to give probability}
\end{cases}
\]

This computation dominates most neural language models operations. The tricky part here is that at the end of the training, we do not actually use the output of the network, instead what we care about is the learned weights \( W \). We will see in the next section that the projected weights matrix \( W \) is actually nothing but the word vectors. In other words, each row (i.e. \( W_i \) where: \( i \in \{1, \ldots, |V|\} \)) is a vector representation of an input word.

**Word Embeddings**

The goal of word embeddings is to capture the semantic and lexical properties of words. This happens by representing words as continuous vectors in a low dimensional space. Word vectors can be obtained either from the internal representation (aka optimized weights) of neural networks [19, 30, 68] or using low rank approximation of co-occurrence statistics [84]. Although it has been shown that the two approaches are closely related [42, 49, 13], the neural networks approach is more common and widely used.

**Word2Vec**

One of the most successful neural network language models was published under a software package known as Word2vec [68, 71]. Word2vec is an algorithmic approach for estimating efficient word
representations from a given corpus. The basic idea behind it is to design a model whose parameters are the word vectors\(^{13}\). Where an objective function is used to iteratively train and evaluate the model at each iteration. Errors evaluation and parameters update is carried out using the well known Backpropagation algorithm.

Word2vec, however, is not a single algorithm. It is a package that comprises two different representations algorithms (Continuous-bag-of-words CBOW and skip-gram), and two different training methods (Hierarchical softmax and Negative Sampling)\(^{14}\).

**General learning model and objective function**

Word vectors are learned by predicting the correct word/context in a sequence of words. Given a large training corpus represented as a sequence of words \(w_1, ..., w_T\), the objective is find the parameters combinations \(\theta\) that maximizes the log-likelihood:

\[
J(\theta) = \sum_{t=1}^{T} \sum_{c \in C_t} \log p(w_c|w_t)
\]

(4)

where \(C_t\) is the set of indices of words surrounding \(w_t\), usually referred to as the context (or window) of \(w_t\).

The probability of observing a context word \(w_c\) given \(w_t\) is parametrized using the word vectors. Given a scoring functions, which maps pairs of (word, context) to scores in \(\mathbb{R}\), a possible choice to define the probability of a context word is the softmax function \(^{51}\).

More precisely, the probability of a surrounding context word \(w_c\) given the center word \(w_t\) is calculated using softmax function as follows:

\[
p(w_c|w_t) = \frac{\exp(w_t^T \cdot w_c)}{\sum_{w_i \in V} \exp(w_t^T \cdot w_i)}
\]

(5)

Where \(V\) is the vocabulary in our corpus. And \(w\) is the vector representation of a word. Note that in the above softmax function it is computationally expensive to evaluate all \(w_i \in V\) “i.e. compute the conditional probabilities of all words” (cost is promotional to \(V\)), therefore the hierarchical softmax is used as an alternative to the full softmax. HS uses a binary tree representation, thus it cuts the evaluation to \(\log(V)\). Similarly, **Negative Sampling** (or Noise Contrastive Estimation

\(^{13}\)The terms word vectors, word embeddings, distributed representation of words, and neural word embeddings are all refer to the same thing; so they are used interchangeably throughout this proposal writing.

\(^{14}\)For more details on this see: Lecture 1 notes of Stanford’s course http://cs224n.stanford.edu
NCE) is proposed as an alternative to hierarchical softmax.

**Skip-gram: A Simple Worked Out Example.** To demonstrate how skip-gram algorithm computes the log-likelihood of the training data, we will walk through a toy example. For demonstration purpose, we will assume a very tiny corpus, which has only one sentence and four vocabulary. Suppose the training corpus contains the following text: “w₁w₂w₃w₄”. And we chose the window size to be = 1 (also called context). The window means that for every target word, we take 2*window_size words as its context, one word from the back and one word from the front, see figure 7 below.

Using the formula 4, the training objective is to maximize the sum of log probabilities for each word $w \in |V|$ and its contexts $C$ as follows:

$$
\begin{align*}
\text{target = } w_1 & & \log p(w_2|w_1) \\
\text{target = } w_2 & & \log p(w_1|w_2) + \log p(w_3|w_2) \\
\text{target = } w_3 & & \log p(w_2|w_3) + \log p(w_4|w_3) \\
\text{target = } w_4 & & \log p(w_3|w_4)
\end{align*}
$$

and by substituting for $p(w_C|w_t)$ with the (full) softmax function in 5, the expression above becomes:
2.2 Language Modeling and Relational Representation

where:

\[ \text{norm}_{w_1} = \sum_{w_i \in V} \exp(w_i^T w_1) = \exp(w_1^T w_2) + \exp(w_1^T w_3) + \exp(w_1^T w_4) \]

\[ \text{norm}_{w_2} = \sum_{w_i \in V} \exp(w_i^T w_2) = \exp(w_2^T w_1) + \exp(w_2^T w_3) + \exp(w_2^T w_4) \]

\[ \text{norm}_{w_3} = \sum_{w_i \in V} \exp(w_i^T w_3) = \exp(w_3^T w_1) + \exp(w_3^T w_2) + \exp(w_3^T w_4) \]

\[ \text{norm}_{w_4} = \sum_{w_i \in V} \exp(w_i^T w_4) = \exp(w_4^T w_1) + \exp(w_4^T w_2) + \exp(w_4^T w_3) \]

And:

\[ w_i = E_{w_i}, \quad E \in \mathbb{R}^{|V| \times d_w} \]

Note that the computation of the \( \text{norm}_{w_i} \) is proportional to \(|V|\). In practice, the size of \( V \) can be as large as \( 10^5 - 10^7 \), therefore, it is often impractical to use the softmax. To address the cost of this computation, two alternative optimization objectives (Hierarchical Softmax and Negative-Sampling) were presented in \[71\].

Learning Relational Features

In machine learning, feature learning refers to the transformation of raw data input to a representation that can be effectively exploited in learning tasks. The choice of data representation can heavily impact the performance of machine learning algorithms [20].

This section emphasizes the importance of data representation, and how it can affects an algorithm’s ability to discover patterns in the input data. Followed by a discussion on the intuition behind deriving relational features motivated by the principle of semantic theory of language.

Feature representation

The learning process in the brain is heavily dependent on its capacity to extract the right set of features from the observed “perceived” world. Thus, the brain learns how to constitute patterns driven by the features it extracts.
Similarly, a learning algorithm learns through discovering patterns in data. To make that happen, the learning algorithm remembers the representation (or features) of the data it processes. The remembering here does not—necessarily—mean explicit memorization of the features. Instead, it is more like gradually-increased confidence about the presence of a certain pattern in the data after a repetitive encounter of the same feature in multiple examples (from the input data). The degree of confidence can then be used to explain or to make judgement on new “un-seen” data; in machine learning this process is known as generalization. Therefore, the better the features (representation of data) the better the learning algorithm can perform and generalize.

In traditional machine learning algorithms, features are hand-designed (by-human). These features are then fed into the model where it learns how to map these data representations to the output. However, for many artificial intelligence tasks, it is difficult to tell what features should be extracted. The goal of learning (or designing) features is to separate what is called the “factors of variation” that explain the observed data. “Factors” refer to the separate sources of influences; often they have no quantity that can be observed and they cannot be combined. For example, in speech recording the factors are speaker’s age, sex, the words they are speaking; and in a car image, the factors are the car’s position in the picture, its color, and the angle of sun brightness.

A major challenge in real-world AI applications is to disentangle these factors, thereafter discard the unnecessary ones. The difficulty lies in that many factors can influence each single piece of the observed data. In the context of the relational features, this challenge lies in the un-quantifiable nature of the relations’ semantics.

One way to solve this problem is to use another machine learning algorithm to discover the representation itself. In deep learning, this approach is known as representation learning. Deep learning solves that problem by expressing representations in term of other simpler ones, i.e. build complex concepts from simpler ones. For example, the concept of a person’s image can be represented by combining simpler concepts such as corners/contours which are in turn defined as edges.

As our target data in this dissertation is textual relational data (i.e. knowledge base triplets), we exploit deep learning approaches to natural language to learn the semantic features of relations. Namely, the approach is neural language models (described in section 2.2). Needless to say, representation of text in general is very important for many real-world applications such as: search, recommendations systems, ranking, spam filtering. In literature, there are different ways to represent text, but
text. The most common techniques as described in [70] include:

- **Local representations**
  - N-grams
  - Bag-of-words
  - 1-of-N coding

- **Continuous representations**
  - Latent Semantic Analysis
  - Latent Dirichlet Allocation
  - Distributed Representations

The approach we follow in this dissertation work, as discussed previously, is based on continuous distributed representations.

### 2.3 Relational Embedding

The representation of raw data is pivotal for the performance of machine learning algorithms. Learning relational features (i.e. relationships between entities) is important for performing machine learning on multi-relational data such as knowledge graphs.

Relational machine learning studies the statistical properties of relational, or graph-structured, data. Research in this area has been active for several years. In general, there are two classes of Statistical Relational Learning (SRL) techniques. Those that use latent variables to capture the correlation between the nodes/edges, and those that rely on the observable properties of the graph to directly capture the correlation. For an in depth discussion on the two techniques applied to knowledge graphs, see [83].

Previously, traditional SRL approaches such as Markov Logic Networks [86] and relational Markov Networks [99] have been dominant in the field. However, these conventional approaches suffer from scalability issues as relational data grow in an unprecedented amount. This has motivated the research community to focus on scalable SRL techniques.

Embedding-based methods have recently emerged as a promising alternative to the conventional SRL. In embedding approaches, relational knowledge of entities and relations is encoded into low-dimensional vector representations. These representations are the corner stone for improving learning in relational models.
2.3 Relational Embedding

We divided the related work on semantic relation embeddings into two categories, those which
develop a general-representational relational embeddings, and those of a task-specific relational
embeddings. We discuss each category in the following two subsections.

2.3.1 Embedding for relation representation

In the past few years, there have been several research dedicated for learning relational embeddings
from knowledge bases for the purpose of knowledge representation. We focus primarily on two of
the top state of the art methods TransE and HolE.

Translating Embeddings (TransE). One of the most notable work in this area was introduced
by [24]. In this model, a relation is represented as a translation vector $r$ from the triplet $(h, r, t)$
such that $h + r \approx t$ figure 8. It predicts the existence of triplets from similarity of the embedding
space (each $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$ has own vector). The assumption is as follows: if $(h, r, t)$ holds, then
the embedding of the tail entity $t$ should be close to the embedding of the head entity $h$ plus some
vector that depends on the relationship $r$.

![Figure 8: TransE represents a triplet relation $r$ as a translation between head $h$ and tail $t$ in the
vector space. (image credit [104])](image)

TransE learn embeddings (take values in $\mathbb{R}^k$) of the entities and relationships by minimizing a ranking
loss function:

$$
\mathcal{L} = \sum_{(h, r, t) \in \mathcal{S}} \sum_{(h', r, t') \in \mathcal{S}'} \left[ \gamma + d(h + r, t) - d(h' + r, t') \right]_+
$$

where $d$ is the dissimilarity measure taken using either the $L_1$ or $L_2$ norm; $\mathcal{S}'$ is a corrupted triplet;
and $[x]_+$ is the positive part of $x$ or zero otherwise. To optimize this loss, the learning algorithm is
carried out using stochastic gradient descent with mini-batch mode.

In contrast to other methods, TransE is appealing due to the fact that it requires very few parameters
and very easy to train. However, TransE has some flaws when dealing with reflexive, 1-to-N, N-to-
1, or N-to-N relations. Other work e.g. TransH and TransR have been proposed to improve the performance of TransE in non-binary relations.

**Holographic Embedding** (HolE). A compositional representations of each triplet. In [82], they employ a circular correlation operation to learn compositional representations of the entire knowledge graph. The method is claimed to be related to the holographic models of associative memory, thus the name Holographic Embeddings or “HolE”. To learn representations of entities and relations, HolE model knowledge base triplets as follows:

\[
Pr(\phi_r(h, t) = 1|\Theta) = \sigma(\eta_{hrt}) = \sigma(r^T(h \circ t))
\]

Where \(\phi\) is the characteristic function: \(E \times E \rightarrow \{\pm 1\}\) which denotes the relation label (\(E\) is the set of all entities). Positive indicates a true relation, i.e. the two pairs of entities \(\{h, t\} \in E\) are part of the relation \(r\), or negative otherwise. The set of all embeddings to be learned by the model is denoted as \(\Theta\). \(\sigma\) is the logistic function, \(\sigma(x) = \frac{1}{1+exp(-x)}\). And \(\circ\) denotes the compositional operator. It is used to create a composite vector representation from \(h\) and \(t\) embeddings by performing a circular correlation operation. The goal is to learn the representation of entities and relations \(\Theta\) that best explain the given triplets dataset \(D = \{(x_i, y_i), i = 1 \ldots N\}\); where \(x_i \in R \times E \times E\) which denotes a triplet, and \(y_i \in \pm 1\). The optimization is carried out by applying Stochastic Gradient Descent SGD to minimize the the logistic loss

\[
\min_{\Theta} \sum_{i=1}^{N} log(1 + exp(-y_i\eta_i))
\]

Other work in literature include several approaches which are either variants of TransE or less powerful than the two methods discussed above. Some of this work include STransE [79], TransR [62], TransH [105], NTN [96] too many parameters and complex model, m-TransH [105] which extends TransH to propose a new framework for multi-fold (or n-ary) relational data, PTransE [103], [109] introduced a hybrid combination of NTN and TransE, SME (linear and bilinear) [23], and finally RESCAL [81].

In Table 2 shows the performance of some of these methods for a link prediction task\(^{16}\).

---

\(^{16}\)Some of the results and experiments are reported from the GitHub repository https://github.com/thunlp/KB2E#evaluation-results
Table 2: Accuracy of current state of the art on FB15k dataset (Raw).

<table>
<thead>
<tr>
<th>Model</th>
<th>MeanRank</th>
<th>Hit@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransE</td>
<td>243</td>
<td>34.9</td>
</tr>
<tr>
<td>TransH</td>
<td>212</td>
<td>45.7</td>
</tr>
<tr>
<td>TransR</td>
<td>198</td>
<td>48.2</td>
</tr>
<tr>
<td>PTransE</td>
<td>216</td>
<td>47.4</td>
</tr>
<tr>
<td>RESCAL</td>
<td>828</td>
<td>28.4</td>
</tr>
<tr>
<td>SME-linear</td>
<td>273</td>
<td>28.8</td>
</tr>
<tr>
<td>SME-Bilinear</td>
<td>284</td>
<td>31</td>
</tr>
</tbody>
</table>

2.3.2 Embedding for relation classification

In this type of work, relational embeddings are developed specifically to tackle a task known as relation classification. The task has been known for many years; and it was formally introduced for research competitions in Task8 of SemEval-2010\textsuperscript{17}. It is focused on semantic relations between two entities in a given sentence. The task is described as follows: given a sentence and two tagged nominals, predict the type relation between those nominals and the direction of the relation [50].

The dataset used in this task is a widely used benchmark for relation classification. It contains 10,717 example sentences (8,000 training and 2,717 testing) annotated with 9 different relation types, with an additional artificial relation labeled as Other, see table 3 for the complete list of types and their frequencies in the data. As an example, the following sentence has a type Entity-Origin relation between the nominals “tea” and “ginseng”:

The cup contained TEA from dried GINSENG.

Traditional machine learning approaches treat this task as a multi-class classification problem, where they apply various techniques in order to achieve high accuracy. Recently deep learning methods have dominated the scene where current state of the art is based on representation learning strategies such as Convolutional Neural Networks CNN and Recurrent Neural Networks RNN.

As a result, the accuracy of relation classification using deep learning models outperformed previous state of the art methods (see table 4). However, the generated embeddings are tailored for the given classification task. Thus, such embeddings are limited to this task, thus, they are not useful as a general relational representation.

\textsuperscript{17}http://semeval2.fbk.eu
### Table 3: Statistics of the benchmark dataset for relation classification.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cause-Effect</td>
<td>1331 (12.4%)</td>
</tr>
<tr>
<td>Component-Whole</td>
<td>1253 (11.7%)</td>
</tr>
<tr>
<td>Entity-Destination</td>
<td>1137 (10.7%)</td>
</tr>
<tr>
<td>Entity-Origin</td>
<td>974 (9.1%)</td>
</tr>
<tr>
<td>Product-Producer</td>
<td>948 (8.8%)</td>
</tr>
<tr>
<td>Member-Collection</td>
<td>923 (8.6%)</td>
</tr>
<tr>
<td>Message-Topic</td>
<td>895 (8.4%)</td>
</tr>
<tr>
<td>Content-Container</td>
<td>732 (6.8%)</td>
</tr>
<tr>
<td>Instrument-Agency</td>
<td>660 (6.2%)</td>
</tr>
<tr>
<td>Other</td>
<td>1864 (17.4%)</td>
</tr>
<tr>
<td><strong>TOTAL:</strong></td>
<td>10717</td>
</tr>
</tbody>
</table>

Table 4: Results of embedding-based relation classification. The symbol † means that one of the classes **Other** (which introduces noise to the dataset) was removed.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Method and Feature</th>
<th>F1 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVRNN [95]</td>
<td>Matrix-Vector Recursive neural network, word embeddings</td>
<td>79.1</td>
</tr>
<tr>
<td>CNN [111]</td>
<td>Convolutional deep neural network, word embeddings</td>
<td>69.7</td>
</tr>
<tr>
<td>FCM[110]</td>
<td>Factor-based compositional embedding, word embeddings</td>
<td>80.6</td>
</tr>
<tr>
<td>CR-CNN [90]</td>
<td>Pairwise ranking with Convolutional neural network, word embeddings</td>
<td>82.8†</td>
</tr>
<tr>
<td>SDP-LSTM [108]</td>
<td>Long Short Term Memory Network</td>
<td>82.4</td>
</tr>
</tbody>
</table>
2.4 Ontology (concepts) and Knowledge Graphs (entities)

The objective of this section is to explain how we interpret the terminology Ontology and Knowledge Graphs in the scope of this dissertation. The idea is not to confuse the reader when seeing words like:

- concept (e.g. Car) vs. entity (e.g. Tesla)
- Ontology has to do with sub-classing (e.g. a sub-class is a class of super-class) vs. knowledge graphs has to do with various relationship with multiple entities
- ontology (i.e. class or concept is-a ) vs. knowledge graphs (i.e. instance)

**Ontology vs. Knowledge Graph**

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Knowledge Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>class or concept (e.g. Car)</td>
<td>entity or instance (e.g. Tesla)</td>
</tr>
<tr>
<td>sub-classing (e.g. a sub-class is-a class of super-class)</td>
<td>various relationships between multiple related entities</td>
</tr>
<tr>
<td>(formally) has a single relationship is-a</td>
<td>any kind of relationship e.g. located_in, has_role</td>
</tr>
</tbody>
</table>

What is Ontology?

Ontology is concerned with the study of being or existence. That is, something being something else. For example, in the sentence Car is an Automobile, we have two concepts Car and Automobile; where the former is a subclass of the latter. In computer science, Ontology is a vital ingredient of AI and Logical systems. A short and formal definition of ontology:

"An ontology is a specification of a conceptualization." Tom Gruber

**Knowledge Base vs. Knowledge Graph**

While both names could be used interchangeably, a knowledge graph is actually a graph-structured knowledge base. However, a knowledge base, on the other hand, is not necessarily a knowledge graph.

- knowledge base (i.e. disjoint facts) vs. knowledge graph (a network of related/interconnected facts)

---

18 For more details on Ontology meaning in the context of computer science, refer to this article: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html
3 Representing Ontology Relationships

We first came across the problem of relational knowledge representation while working with semantic web ontologies to represent relationships i.e. representing the semantic relations between two nominals.

The following section is based on our published work [9].

3.1 Supporting Part-Whole Relations with OWL and Description Logics (DLs)

In this section, we present our initial work on how we applied Semantic Web Tools and Technology to extend the representation support to more than one type of relationship in ontologies.

Through this work, as it turned out, we are encountered with the problem of representing the semantics of the relationships and reasoning over them. We also tried to describe the role of ontologies in knowledge representation and reasoning. The flow of the following subsections and paragraphs intends to convey the idea that ontology is an essential building block for intelligent systems.

The correct representation of hierarchy in ontologies is a crucial requirement to infer the true knowledge. OWL is a widely used language for building web ontologies. However to represent partonomic relations, unlike the generalization-based taxonomy (is-a relations), OWL does not provide any built-in primitives neither a straight-forward procedure for part-whole relations [85]. However, although, OWL-DL contains sufficient power to describe such relations, it still requires extra work to make full use of this power. The current approach (best practices recommended by w3.org) uses that power through manual implementations of the underlying steps; which could introduce an error-prone process and may cause confusion in some cases.

In this paper, we propose a simplified approach to represent part-whole relations in a similar way to subClassOf. Our method utilizes OWL-DL’s power through automated generation of the relation’s restrictions. We make use of OWL’s annotation properties to capture the relations constraints, then apply our method to automatically generate the equivalent OWL-DL representations. The evaluation of our representation shows that the classification results and the inferred model (safe reasoning) are the same as those of the current standard approach described in [85].
3.1 Supporting Part-Whole Relations with OWL and Description Logics (DLs)

Knowledge, Knowledge Representation, and Ontology

Grosan [46], defines knowledge in a generic way, as “information (which can be expressed in the form of propositions) from the environment.” He, also, defines knowledge representation as “symbols used to represent the propositions.” Based on that, Grosan describes knowledge representation and reasoning as “the manipulation of symbols encoding propositions to produce representations of new propositions.”

Gruber [47], defines Ontology in short as “a specification of a conceptualization.” Ontology, as an important ingredient in Artificial Intelligence field, is one of the common way used for representing knowledge. In fact, ontology is an essential component for many intelligent systems as well, for example semantic web, automated medical evaluations, IBM-watson, personal assistant and knowledge navigators “e.g. Siri in Apple’s mobile devices”, and many others. In particular, ontology usage is gaining increased interest in biomedical and military research.

However, representing partonomy relations in domain specific ontologies is a crucial prerequisite for automated reasoning. For example, in the biomedical domain, coping with only ontological dependencies between wholes and their parts may not be sufficient but rather the more important is to capture the reasoning patterns which underlie the propagation of the partonomic hierarchies [92].

This paper is organized as follows: Section 2 introduces the problem of part-whole in knowledge representation, presents a literature survey of the proposed types and semantics of such relations, and discusses the reasoning complexity. Section 3 describes our proposed approach with examples and the evaluation methods. Section 4 discusses related work followed by the conclusions and future work in Section 5.

Related work

Various work in literature were proposed to address the issue of partonomy problem in ontology. We can distinguish between two types of work dealing with this problem. In the first type of research, the focus were more on representation aspects; where in the second type, the work focused more on the reasoning aspects.

19 In this paper, we discuss ontology from a computer science point of view not from a philosophical perspective.

20 Basic Formal Ontology: http://ifomis.uni-saarland.de/bfo/
In [92], Schulz et al. try to solve the biaxial (reflecting both a taxonomic “is-a” and a partonomic “part-of” hierarchy) structure in biomedical ontology by scaling down part-whole reasoning to subsumption-based taxonomic reasoning in the expressivity level $ALC$ of description logics.

In another medical related work [93], Seyed et al. discussed the current approaches for representing partonomic relations in SNOMED CT. They investigated the Structured-Entity-Part “SEP” triple approach which is related to DLs at different levels of expressivity.

The Dilemma of Partonomy (Part-Whole) Relations

The complexity in representing part-whole relationships is not about how to represent them. The main problem lies in how to reason over these relationships when they are represented in a certain structure. In fact, reasoning (inference) is the ultimate goal for using ontology to represent knowledge for many applications, and the semantic web is no exception. Therefore, the format of the structure should be a machine-readable so that intelligent agents can comprehend that representation.

Unlike the generalization-based taxonomic reasoning (i.e. $C$ is SubClassOf $B$, $B$ is a SubClassOf $A$, thus, $C$ is a SubClassOf $A$), there are different types of part-whole relationships and each type may have a different level of transitivity and associativity. This fact affects ontology building in two ways; First there cannot be a single form to represent such relations, and as a result. Second; increased complexity in partonomic reasoning.

Thus, any existing or proposed representation mechanism has to take into account these differences in order to enable intelligent agents to infer the true intended knowledge.

Types of Relations and Part-Whole Relations

The study of part-whole relationships is an entire field of study by itself called “mereology” [85]. Although these relations may at first sound as a well-defined piece of knowledge, in practice, however, the case is quite different. A consensus on the semantics of these relations yet to be achieved, which contributes to the complexity of both representing and reasoning for these relations. Therefore, it is important to establish common grounds about the conveyed semantic through each different part-whole relation type.

In [26], Barchman analyses the semantics of the IS-A links in semantic networks through distinguishing two general types of the meanings for this inheritance link. In order to interpret the relation
3.1 Supporting Part-Whole Relations with OWL and Description Logics (DLs) 

links, he describes two type of nodes 1) generic “internal nodes”, the relation between nodes of this type is described as generic/generic relations, in description logics terminology, this can be defined as the Terminological Box “TBox” relations. 2) individuals “leaves”, here the relation between an individual and a generic nodes is described as generic/individual relation in description logics terminology, this can can defined as the Assertional Box “ABox” relation.

The intent of generic/generic relations is usually that one is somehow related to, but less general than, the other. The kind of uses in this relation include (1) subset/superset, (2) generalization/specialization, (3) a kind of, e.g. a camel is a kind of mammal, (4) conceptual containment, called the is-a of lambda-asbraction, e.g. a triangle is a polygon, (5) role value restriction, e.g. the trunk of an elephant is a cylinder 2 meters long, and the (6) set and its characteristic type.

While in the generic/individual relations, the intent is that the individual is describable by some general descriptions. This type of relations is commonly called “instantiation”. The kind of uses in this relation include (1) set of membership, e.g. Clyde is a camel, means that Clyde is a member of the set of camels. (2) predication, (a predicate to an individual) e.g. if the generic is Camel and the individual is Clyde, this relation conveys that Camel(Clyde), (3) conceptual containment, e.g. the relationship between “King” and “the king of Saudi Arabia”. Generic is used to construct the individual description, (4) abstraction, e.g. the eagle is an endanger species.

As an attempt to explain the semantics of meronymic relations, [106] developed a taxonomy for part-whole relations to explain the ordinary English-speaker usages of part-of. Their classification yields six types of part-of relations. The types are 1. component-integral object, pedal-bike, 2. member-collection, ship-fleet, 3. portion-mass, slice-pizza, 4. stuff-object, steel-vehicle, 5. feature-activity, paying-shopping, and 6. place-area, Manhattan-New York.

The six types are differentiated from one another by three properties that hold associatively between parts and wholes, functional: where the parts play the functional role; homoeomerous: where the parts are similar to the whole and to each other; separable: where the parts are disconnected from the whole [7].

In practice, [85] distinguishes two categories, simple and complex, of part-whole relations when they demonstrate how to, formally, represent part-whole relations in OWL. The first, simple part-whole relations, denotes most of the straightforward cases, i.e. , the finger is part of the hand, and, an engine is part of a car. The second category describes the complex part-whole cases that should not be confused with previous types. The cases include types such 2) containment i.e. the chair is
contained in the room, does not mean that the chair is part of the room 3) *membership* i.e. a faculty member is part of the committee, here membership is not transitive. 4) *connections and branches* i.e. the tributary, it is connected to the river, is not part-of the river. Likewise, a light bulb when connected to electricity does not mean it is part of it. 5) *constituents*, controversially, this kinds distinguishes between the clay and a statue made of clay. Here it described as the clay constitutes or is constituent of the statue.

In addition to the aforementioned, many other types of part-whole relations have been proposed in the literature. However, many of them may lack a formal specification to convey clear semantics to them. So to tackle this problem, [54] proposed a formal taxonomy model (Fig. 9) for the (mereological and meronymic) part-whole relations. The model, which is based on various part-whole relations that have been introduced and/or discussed in the literature, presented as a way of dealing with part-whole relations in conceptual data models (and domain ontologies).

![Figure 9: Keet’s Taxonomy of basic mereological and meronymic part-of relations. Dashed lines indicate that the subtype has additional constraints on the participation of the entity types; ellipses indicate several possible finer-grained extensions to the basic part-whole relations.](image)

**Reasoning Complexity**

Developing a powerful, robust, and reliable domain ontology that support large-scale of formal reasoning is a key requirement for intelligent systems. However, the diverse forms of partonomies (as discussed above) introduces a real challenge toward realizing that requirement. Thus, and as part of the efforts to remedy this obstacle, two aspects of reasoning on part-whole relations have received special attention in the literature *Transitivity* and *Taxonomic Reasoning* [48].

In *transitivity*, it has been long debated whether the part-whole relation’s transitivity holds for adequate reasoning or not, that is, e.g. if concept $x$ is *part-of* concept $y$ how far in the hierarchical levels could this relation hold for the upper concepts. In [106], they argue that part-whole relations should be considered transitive as long as they keep a *single sense of part*. From this perspective, the general part-whole relation is not transitive, but each of its distinct sub-relations are. However, generally
when more than one single-sense involved in the chain of part-whole sub-relation, transitivity no longer holds. Thus, [48] argue that this problem “cannot be solved at the level of the axiomatic definition of knowledge-representation languages and, for example, the transitivity operators with which they can be equipped.”

For taxonomic reasoning in partonomies, two patterns that crucially depend on part-whole relations were discussed in the literature. The first one relates to role propagation in partonomies. Generally, if two relations, R “part-of” and S “sub-relation of part-of” (that is, \( S \subseteq R \)), are given and we have \( xRy \) and \( ySz \) then the following implication holds:

\[
xRy \land ySz \Rightarrow xRz
\]

The second pattern is that the above framework allows concept specialization in partonomies. Generally, this pattern phrased as follows: for two relations \( R \) and \( S \) such that \( S \subseteq \text{part-of}, \) then:

\[
xRy \land wRz \land ySz \Rightarrow x \text{ is-a } w
\]

We notice that the second reasoning pattern is actually a special form of the first one [48].

**Expressivity and Reasoning complexity**

The trade-off relation between expressivity and reasoning complexity is another important issue to take into consideration when developing ontology representations.

For instance, ontological knowledge can be represented in different profiles of OWL (e.g. using different levels of description logics expressivity such as \( \text{AL, ALC, SHIF, SHOIN} \)). Clearly, the more detailed our ontological representation the more knowledge we can infer; however, as a result, the cost of this will be increasing level of complexity for the available reasoning tools. On the one hand of this issue there are large ontologies like SNOMED CT, expressed with expressivity logics such as OWL EL; on the other hand, the issue does not end with the relatively expressive OWL DL. Since some of the true knowledge might be lost when ignoring the details in the representation, it is a common practice to intersperse OWL with FOL axioms in the comment or annotate them. Although these axioms will be ignored by reasoners [56].

In [93], they investigated the representation of the part-whole relationship in SNOMED CT. As part of their analysis, Seyed and Rector et. al. intend to inform the SNOMED CT community to take the trade-offs issue into account for future decisions about the representation of their anatomical taxonomy.
Our Approach

We introduced a straightforward method that exploit OWL’s annotations, and used in RDF/XML syntax, to define classes for parts in partonomic relations. The goal of our approach is to provide a simplified OWL primitive for describing part-whole relations in the same way used for describing subclass relations.

Our approach is based on the representation discussed in [85]. We further simplify the mechanism that Rector et al. described for representing part-whole in OWL. In the third representation pattern “Defining classes for Parts” (described in [85]), they demonstrated how to formally represent TBox part-whole relations in OWL. Their technique involves three separate tasks prior to producing the preferred partonomic ontology. Each task could comprise more than one step. Some of the steps may not be intuitive which introduce additional confusion to the ontology developer. The three tasks in general are:

1. defining a property “role” with preferred restrictions
2. defining an aggregating class for parts “classPart”
3. extending the part class to cope with the restrictions of the role relationship.

This approach [85], is currently recognized (by www.W3.org) as the standard method for representing part-whole classes in OWL. Therefore, in this paper, we will refer to that technique as the current approach and we will refer to our technique as the proposed approach.

In the proposed approach, we integrate all the three steps mentioned above in a single RDF/XML line (See listing 3) and supports transitivity and cardinality declarations as well. The representation reasoning of the proposed approach yields the same inference as the current approach.

In addition, we provide a simple technique for visualizing the final inferred ontology model which include the partonomic relations from our simplified approach.

OWL: Part-Whole Representation Problem Revisited

The problem of part-whole relations in OWL is that there is no built-in primitive for representing such relations. However, OWL does provide sufficient expressive power to support most of the partonomic relations [85]. But in order to utilize this support, an ontology developer has to go through some extra work “work-arounds” to successfully express the intended representation. This

extra work, however, includes “manually” conducting some underlying steps where they can be automated.

**Scenario**

To explain this better, and to demonstrate both current approach and the proposed approach solutions, we will use a simple use-case scenario with *is-a* and *partOf* relations. We show the current approach mechanism to define the partonomic relation in OWL. Thereafter show our proposed approach and compare the two. The following scenario example includes both types of relations *generalization-based* and *partonomic-based*. Assume we would like to, formally, represent the piece of knowledge visualized in Figure 10, that is to use OWL to define:

- *Vehicle* class as a generalization of *(is a)* *Car* class, and
- *Engine* class as a *part of* *Car* class.

![Figure 10: Simple Scenario For PartOf and IS-A relations](image)

To represent the is-a relation, we can, simply use the predefined *keyword* *(subClassOf)* as shown in listing 1.

**Algorithm 1** Defining is-a relation in OWL

```xml
<owl:Class rdf:about="NS#Vehicle">
  <rdfs:subClassOf rdf:resource="NS#Car"/>
</owl:Class>
```

However, there is no such straight-forward way *(keyword)* for defining the part-of relation. The reason for this is that: (1) partonomic relations are different and there could be multiple forms for such relations as discussed earlier. (2) there are some constraints e.g. transitivity, associativity, and cardinality need to be imposed to specify the relation’s restrictions.

Hence, we could utilize the power of OWL-DL to express this relation with its constraints. The process for doing so may vary depending on the expressivity level involved in the relation. However, in this example we consider the direct and simple form of part-of relation as depicted Fig.10.

Thus in order to define the *part of* relation using OWL, the required process proceeds as follow:
3.1 Supporting Part-Whole Relations with OWL and Description Logics (DLs)

- Create an objectProperty `partOf` (with specifying whether it is transitive or not) to represent the relation between the two classes.

- Create a new part-aggregating class `carPart` to represent the type of the part class. Then use OWL-DL to:
  - make it an `equivalentClass` of the restriction `partOf some Car`.
    
    e.g. `carPart ≡ ∃ partOf Car`

- Extend the part Class `Engine` to cope with the constraints. By using OWL-DL to:
  - make it a `subClassOf` of the restriction `partOf some Car`.
    
    e.g. `Engine ⊑ ∃ partOf Car`

In this way, the `carPart` will denote the aggregation class point where the car parts classes e.g. `Engine` become part of the class `Car`.

Although our intention is to describe one well-defined piece of knowledge (that is, `Engine is a part of the Car`), the Current Approach for defining that relation implements each of the tasks “mentioned above” separately and independently. This could be a long, tedious, and error-prone process. The outcome of these independent tasks is shown in listing 2.

**Algorithm 2** Current approach for defining Engine as part-of Car

```
<owl:ObjectProperty rdf:about="&part;partOf">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>

<owl:Class rdf:about="ns#Engine">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="ns#partOf"/>
            <owl:someValuesFrom rdf:resource="ns#Car"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="ns#CarPart">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="ns#partOf"/>
            <owl:someValuesFrom rdf:resource="ns#Car"/>
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

The Proposed Approach, in the contrary, tends to make this description as simple and abstract as possible. Where we combine all the three tasks in a single integrated task. This will 1) reduce the amount of work needed, and 2) remedy the complexity level involved in the process, hence 3) minimize the possibility of making errors. Listing 3 illustrates our simplified approach. As appears from the listing, we introduce the annotation property `relation` as a keyword for describing any
part-whole relations. Thus clearly, the approach could be applied with any type of part-whole relations not only part of.

**Algorithm 3 Proposed approach** for defining Engine as part-of Car

```
<owl:Class rdf:about="NS#Engine">
  <relation:partOf transitive="yes" rdf:resource="NS#Car"/>
</owl:Class>
```

The idea of this approach Fig. 11 is to simulate the partonomic relation “using annotation property” with specifying its constraints “using attributes” in one line. Then we apply our method to automatically 1) extract these information, thereafter 2) map them to the matching specifications, where it will then 3) generate the proper OWL elements.

```
Simulated Relations  Using OWL Annotations

↓

Transformation  Relation ↔ Elements Mapping

↓

Generate Final layout  Standardized RDF/XML
```

Figure 11: Proposed Approach Conceptual Work Flow Model

In the automated generation step, we produce a new name for the aggregating class by concatenating the name of relation (e.g. partOf) with the name of whole class (e.g. Car). When concatenating the two strings, we use an underscore (e.g. partOf_Car) to keep the two separable for future distinguishing.

```
Simplified representation of a relation

Parse the simplified relation to extract its constraints

Automated Mapping

extend current class to be subClassOf parent

generate an objectProperty of the relation

generate part-aggregating class as an equivalentClass of parent
```

Figure 12: Conceptual Model: The Mapping Stages of Our Method
Automated Generation Steps

To further elaborate on this method, we provide step by step examples of how the method produce the final OWL layout. Figure 12 illustrates the general outlines of mapping stages. The input will be a simulated part-whole relation (like the one in listing 3), so the method starts by parsing that line to extract the relation’s 1. part class, 2. whole class, and 3. specifications (e.g. transitivity, and existential). From there, the part class will be extended as a sub class of the whole class (listing 4). The property object will be generated as shown in listing 5. And finally the aggregating class will be generated as appears in listing 6. This process will happen continuously as long as the input OWL file contains the simulated relations.

Algorithm 4 Automated mapping stage 1

```xml
<owl:Class rdf:about="ns#Engine">
<!-- Auto generated mapping -->
<rdf:s:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="ns#partOf"/>
<owl:someValuesFrom rdf:resource="ns#Car"/>
</owl:Restriction>
</rdf:s:subClassOf>
<!-- End mapping -->
</owl:Class>
```

Algorithm 5 Automated mapping stage 2

```xml
<!-- Property -->
<!-- Auto generated mapping -->
<owl:ObjectProperty rdf:about="ns#partOf">
<rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>
<!-- End mapping -->
```

Algorithm 6 Automated mapping stage 3

```xml
<!-- Auxiliary Class -->
<!-- Auto generated mapping -->
<owl:Class rdf:about="ns#partOf_Car">
<owl:equivalentClass>
<owl:Restriction>
<owl:onProperty rdf:resource="ns#partOf"/>
<owl:someValuesFrom rdf:resource="ns#Car"/>
</owl:Restriction>
</owl:equivalentClass>
</owl:Class>
<!-- End mapping -->
```

Approaches Compared

To compare our approach with the current approach, we use the *Car*\(^{22}\) ontology example discussed in [85]. However, since our method currently deals with ontologies of description logics type \(AL\) and

\(^{22}\) part.owl, example2.owl, and example3.owl
map them to \textit{ALE+}, we modified the \textit{Car} ontology example to exclude the \textit{ALEHI+} part namely:

- Inverse properties
- Sub-properties

As a way to statistically compare the two approaches, we use the \textit{Car} example to collect the ontology metrics from both current and proposed approach, see our complete simulated approach listing \ref{7}. Then we compare the two where we consider the different values of the metrics only. The results are show in Figure \ref{13}. We, also, notice that the difference between the two methods gets larger with big ontologies. The \textbf{important thing} to mention here is that both ontologies provide the exact level of representation expressivity. Thus, the reasoning underlie these two representations is the same. Figure \ref{14} show the final inferred model from the representations.

This simplified way will help ontology developers to focus on the logical hierarchy of the ontology, as a result of reducing the work needed to produce the auxiliary \textit{partClasses} and their \textit{accompanying constraints} separately.

![Figure 13: Ontology Metrics Comparison: Current Approach vs. Proposed Approach](image)

**Evaluation Method**

In order to evaluate our approach, two things need to be tested when we simulate and generate the final ontology layout:

1. the format of generated ontology from our mapping process must be syntactically validated and formed correctly “well-formed”, and
2. the inference from the representation must convey the true knowledge, e.g. correctly express the knowledge while support ontology reasoners.

For validating our generated ontology, we use saxParse engine (built-in in the commonly used ontology editor Protege 5.0) for verifying the well-formedness of our ontology at each mapping stage throughout our experiments.
3.1 Supporting Part-Whole Relations with OWL and Description Logics (DLs)

![Diagram](image)

(a) The Inferred Model of Our Transformed Representation

![Diagram](image)

(b) The Inferred Model of The Current Approach

Figure 14: We use HermiT 1.3.8 reasoner (built-in protege) to evaluate the representation reasoning of both (a) our approach (b) the current approach for ontology example 3. And the results (as anticipated) are the same.

While for evaluating the representation reasoning and to ensure a correct inference, we used HermiT 1.8.3. reasoner (built-in reasoner in Protege 5.0).

**Future Work**

As the introduced approach simplifies the process involved in describing partonomic axioms, developing OWL ontologies becomes even simpler than what we usually experience. Specially, when we apply this approach for building a domain specific ontology. Therefore, we are most excited to further exploit the simplified approach for Semi-Automatic Ontology Building. Actually, we have already started this experiment. Where, we acquire knowledge bases in the form of triples, then we use our dictionary data structures to represent the triples as input for our simplified approach.

Also, we are going to extend the support of our approach to include mapping to other levels of OWL-DL, e.g. inverse and subProperties (ALEHI+).

We are also planning to interface our module with Protege through a plugin. Another thing we are looking into is the possibility of conversion to other OWL formats (e.g. Manchester, Functional OWL).
Conclusion

As intelligent systems (AI technologies in general) become more dependent on the ontological representations for eliciting knowledge, the accuracy of ontology hierarchy contributes heavily to the success of those systems. The building blocks of this hierarchy are the taxonomic (generalization-based) relations and the partonomic (part-whole) relations. Although the former has a well-defined underlying knowledge, thus, a clear representation, the latter is still lacking.

However, despite the ongoing non-consensus about how to represent the underlying knowledge of partonomic relations, we believe that providing a clear and less ambiguous procedure to denote such relations will surely lessen the reasoning confusion.

In this paper, we have introduced and demonstrated our simplified approach for representing part-whole relations in OWL. Where we utilized OWL’s annotations properties to represent part-whole relations in a manner inspired by subClassOf.

By doing so, we believe that our approach contributes to (1) reduce the amount of work needed for building ontologies, (2) remedy the complexity involved in representing part-whole relations, (3) minimize the possibility of making errors throughout the process, and in the future to (4) provide support to Semi-Auto Ontology Building.

Figure 15: Visualization of The Simplified Ontology in Listing 7: Using Our Module We Visualize The Inferred Model Directly From The Simplified Representation
Algorithm 7 Our Complete Simplified Representation For Car ontology (Example 3)

```xml
<owl:Class rdf:about="ns#Carburator">
  <relation:partOf rdf:resource="ns#Engine"/>
</owl:Class>

<owl:Class rdf:about="ns#Crankcase">
  <relation:partOf rdf:resource="ns#Engine"/>
</owl:Class>

<owl:Class rdf:about="ns#Engine"/>

<owl:Class rdf:about="ns#Wheel"/>

<owl:Class rdf:about="ns#Carburator"/>

<owl:Class rdf:about="ns#Reflector"/>

<owl:Class rdf:about="ns#Engine">
  <relation:partOf rdf:resource="ns#Car"/>
</owl:Class>

<owl:Class rdf:about="ns#Wheel">
  <relation:partOf rdf:resource="ns#Car"/>
</owl:Class>

<owl:Class rdf:about="ns#Headlight">
  <relation:partOf rdf:resource="ns#Car"/>
</owl:Class>

<owl:Class rdf:about="ns#Carburator"/>

<owl:Class rdf:about="ns#HeadlightBulb"/>

<owl:Class rdf:about="ns#HeadlightBulb">
  <relation:partOf rdf:resource="ns#Headlight"/>
</owl:Class>

<owl:Class rdf:about="ns#Reflector">
  <relation:partOf rdf:resource="ns#Headlight"/>
</owl:Class>

<owl:Class rdf:about="ns#Headlight"/>

<owl:Class rdf:about="ns#Crankcase"/>
```
3.2 Supporting Multi-Type Relations with Machine Learning

3.2.1 Employing embedding methods to represent Relationships and Entities:

In our earlier work, we tackle this problem by relying on Web Ontology Language to encode the entire knowledge of a given domain. However, hand-coding knowledge is challenging. It is not possible to capture all the different types of relationships and associations between concepts. As a result, this approach suffers when it comes to scalability, adaptability, and maintainability. So we had to turn our focus to a more scalable approach, that’s when we decided to use machine learning to approach this problem. Although the two approaches are relatively unrelated, in this dissertation we describe our experimentations with both approaches.

Hand-coding knowledge. In our previous work, we relied on semantic web tools such as OWL and RDF to build ontology models such that it can be exploited by an intelligent computer program (e.g. reasoner) to perform tasks such as knowledge inference.

These tools are suitable for representing generalization-based (i.e., IS-A relations) relational knowledge. Such that it can describe the sub-hierarchical associations between entities (e.g. “Car is a Vehicle”, “Vehicle is an Automobile”, and so on). However, things can get a bit complex and tricky as we encounter relation types other than IS-A. The description level in OWL ontologies (i.e. the expressiveness) is very crucial for conveying the underlying knowledge of the relations and associations between entities.

To this extent, OWL comes in variant description-logic-based sublanguages (OWL Lite, OWL DL, and OWL full) for facilitating the representation of complex associations and different semantic relation types (e.g. part-of and has-part). Description Logics empower OWL with the required capability to formally specify different kinds of semantic relations. Nonetheless, this comes with its cost.

Utilizing OWL’s capability to build a certain ontology requires a close expertise supervision and multiple manual steps. For example, to construct a gene-related ontology, a domain expert (gene specialist in this case) needs to specify the semantic relations between genes and their interactions. Ontology developers are then to perform a series of OWL specification steps in order to translate these semantic relations into formal rules. As an attempt to mitigate some of the overwhelming steps for building ontologies, we had proposed a simple and automated approach [9] to reduce some

---

The Ontology Web Language or OWL is a Semantic Web language designed to represent rich and complex knowledge about things, group of things, and relations between things, see https://www.w3.org/OWL/.
of the steps required for specifying *part-whole* relations in OWL-DL ontologies.

However, beside being a tedious task, the ontology building process becomes challenging as the knowledge domain expands and relation types increase. Hence, with the inherit complexity of the world’s rules, domain experts and ontology developers alike struggle to devise formal specifications to accurately describe the world with minimum complexity.

**The search for a scalable and future-proof solution.** As a result of these limitations, we had to investigate other approaches that can be better at handling scalability and have the ability to self-learn the rules of semantic relations and their associations. Therefore, in the second phase of our research we moved on to focus more on machine learning methods. Such that, our objective is to develop a supervised model that can uncover the underlying structure of semantic relations and capture their association rules through learning from examples. Knowledge base (or knowledge graph) data is the example in this case.

### 3.2.2 Visualizing entities and their relationships

During our early exploratory analysis of the data, we needed a quick way to visualize knowledge base triplets in a directed graph fashion. There are plenty of software and tools that can be used to build and visualize graphs. However, most often in such tools one would need to build each node separately and then, in a subsequent step, add edges between the related nodes. This is not quite efficient for our case specially with the large collection of *(head, relation, tail)* triplets. So instead, we wanted a tool that we can use to build and visualize KB graphs in batches just by feeding the triplets as plain statements. To do that, we have created a small Python module which we called PyGraph\(^\text{24}\). PyGraph is essentially a customized wrapper around a popular graph visualization software called “Graphviz”\(^\text{25}\). With PyGraph we can build knowledge graphs from raw triplets with specifying the delimiter that bisect the triplet’s *head*, *relation*, and *tail*. The relational statements can be supplied to an instance pygraph class either from the command line or within-module by streaming them from a file.

Say for example we have the file “*molecule.txt*” which contains a knowledge base fragment about the entity “Molecule” as follows:

\[\text{__radical\_NN\_1 \hspace{0.5cm} _part\_of \hspace{0.5cm} _molecule\_NN\_1}\]
\[\text{__physics\_NN\_1 \hspace{0.5cm} _member\_of\_domain\_topic \hspace{0.5cm} _molecule\_NN\_1}\]
\[\text{__molecule\_NN\_1 \hspace{0.5cm} _has\_part \hspace{0.5cm} _atom\_NN\_1}\]
\[\text{__unit\_NN\_5 \hspace{0.5cm} _hyponym \hspace{0.5cm} _molecule\_NN\_1}\]

\(^{24}\text{See code and details at: https://github.com/iamaziz/pygraph}\)

\(^{25}\text{http://www.graphviz.org/}\)
3.2 Supporting Multi-Type Relations with Machine Learning

We can build the complete knowledge graph of this fragment by using few lines of code, e.g.

```python
>>> from pygraph.dgraph import PyGraph
>>> g = PyGraph()
>>> g.file_relations("molecule.txt")
>>> g.draw_graph()
```

Which would instantly generates the directed graph shown in figure 16.

![Graph](image)

Figure 16: An example fragment of WordNet KB in a graph generated by our PyGraph wrapper.

**Exploring word embeddings.** As part of getting acquainted with neural language modeling technique, we have experimented on applying word embeddings to a real world problem. Where we build and train a model for detecting sentiment in Arabic text \[10, 8\] using neural word embeddings alone as the primary source of features.

In order to train the sentiment classifiers, we had to first build word representation model. For that purpose, we compiled a corpus of around 190 million words where we use them in our CBOW language model to generate word vectors for the corpus vocabulary. The results of our experiments have proven to us that feature representations using neural language model are superior to state of the art manual feature extraction methods.\[26\]

### 3.2.3 Experimental observations

In this section, we present our preliminary work on this problem (i.e. relation embeddings); which was mainly focused on experimenting and testing the existing approaches in literature. Our completed work is introduced in chapter 5.2.

\[26\]See embeddings and related code at https://github.com/iamaziz/ar-embeddings
3.2 Supporting Multi-Type Relations with Machine Learning

**Initial observations.** As we saw in algorithm 8, the model’s objective is to learn embeddings for the entities $\mathcal{E}$ and relations $\mathcal{R}$ that best describe the training examples $\mathcal{D}$ through minimizing the loss function. By the time of this writing, the final model is not concluded yet. However, we carried out several preliminary runs (on WN18 dataset) during the parameters tuning and score function search. And we have noted few observations from those runs.

We noticed, as pointed out by [24], that smaller embedding size (e.g. $k = \{20, 50\}$) tends to perform quite better in compare to larger dimensions (e.g. $k \geq 100$), see figure 17. Likewise, a larger batch size (e.g. 400-500) performs relatively better than smaller ones (e.g. 100), see figure 18.

We also visualized the learned embeddings of the 18 relation types in a 3D vector space. Figure 19 displays one of the relations (_part_of_) compared to its most similar neighboring relations.

![Figure 17: Ranking loss with different embedding size. From top to bottom embedding size $k = \{100, 50, 20\}$](image17.png)

![Figure 18: Slight improve in ranking loss after increasing the batch size from 100 (orange) to 400 (blue)](image18.png)

**Tools used.** In all conducted experimentations we use Python as the main language with its rich scientific ecosystem libraries such as NumPy (for numerical computation) and Pandas (for data processing and analysis). To build and train the neural language model, we use TensorFlow framework [1] through its Python API27. As an open source framework, TensorFlow is used primarily

---

27 [www.tensorflow.org/api_docs/python/](www.tensorflow.org/api_docs/python/)
3.2 Supporting Multi-Type Relations with Machine Learning

Figure 19: Visualizing relation embeddings in a 3D vector space model (dimensionality reduction is based on t-SNE). In the sample: appears top 5 nearest neighbors (using cosine-similarity) of the relation \_part\_of.

for constructing neural networks in computational graphs manner that support both GPU and CPU computation. To easily inspect, understand, debug, and optimize the built model, TensorFlow provides a nice web application suite called TensorBoard\textsuperscript{28}. We use TensorBoard to visualize the learned embeddings and monitor training errors.

The great thing about TensorFlow, and other deep learning frameworks such as PyTorch\textsuperscript{29} and Theano\textsuperscript{30}, is their productivity for quick model construction and automatic differentiation feature. They can easily calculates all derivatives necessary for running an optimization model such as stochastic gradient descent.

\textsuperscript{28}www.tensorflow.org/get_started/summaries_and_tensorboard

\textsuperscript{29}http://pytorch.org/

\textsuperscript{30}https://github.com/Theano/Theano
4 Embedding-Based Representation: Learning Embeddings for Relational Data

4.1 Building Embeddings From Raw Text

We consider knowledge graphs as a form of natural language. In other words, the triplets are written in linguistics format, so they possess the linguistic characteristics that apply to normal text. The goal of this section is to demonstrate the capability of neural embeddings in capturing the semantic and syntactic features of the language.

4.1.1 Embeddings to replace hand-crafted feature extraction in text

The following is based on our published work [10, 8].

Building and applying embedding model to detect sentiment in Arabic text

Manual feature extraction is a challenging and time consuming task, especially in a Morphologically Rich Language (MRL) such as Arabic. In this experiment, we rely on word embeddings as the main source of features for opinion mining in Arabic text such as tweets, consumer reviews, and news articles. First, we compile a large Arabic corpus from various sources to learn word representations. Second, we train and generate word vectors (embeddings) from the corpus. Third, we use the embeddings in our feature representation for training several binary classifiers to detect subjectivity and sentiment in both Standard Arabic and Dialectal Arabic. We compare our results with other methods in literature; our approach—with no hand-crafted features—achieves a slightly better accuracy than the top hand-crafted methods. To reproduce our results and for further work, we publish the data and code used in our experiments.\footnote{\url{https://github.com/iamaziz/ar-embeddings}}

Introduction

Sentiment analysis is a very common task in Natural Language Processing (NLP), where the goal is to determine the attitude or feeling conveyed in some text. With the surge in microblogging and
similar services, opinionated posts and texts are flooding the Internet. Hence, sentiment analysis has gained much attention and is in demand for many applications (e.g. business analytics) because of its simplicity and efficiency. Various products are being developed around users’ opinions ranging from consumer reviews to reactions surrounding political events.

Although English has been the target language in most sentiment analysis research, recent efforts extend the focus to other languages such as Arabic. Basic machine learning techniques—as simple as Naïve Bayes—have been used to achieve baseline results [16, 75]. However, these systems require lots of feature engineering work prior to applying any machine learning method.

Most Arabic sentiment analysis systems still rely on costly hand-crafted features, where features representation requires manual pre-processing in order to obtain the preferred accuracy. For example, Mourad and Darwish (2013) report that POS tagging and word stemming have major effect in improving their sentiment classification result. Morphology-based features have, also, been shown to improve the system’s performance [4]. A manually-prepared list of emotion words “polarity lexicon” is another requirement in such systems. And not to mention the efforts spent on handling random sentence structures in dialectal Arabic.

In this work, we present neural word embeddings as an alternative for such hand-crafted features in Arabic sentiment analysis. We embed Arabic words in a continuous vector space. This allows us to represent sentiment features as dense vectors instead of the conventional sparse representations. We consider the sentiment task as a standard binary classification problem, thus we discriminate only between either positive/negative or subjective/neutral sentiment.

The contributions of this paper include:

- Employing distributed word representations to embed sentiment features as dense vectors in Arabic sentiment analysis; where we achieve a significant performance.
- An Arabic corpus that we have built carefully from various text collections.
- A pre-trained Arabic word embeddings generated from the aforementioned corpus.
- A labeled (positive/negative) Arabic twitter dataset that we gathered from several published datasets and refined them into a larger dataset.

**Related Work**

We are not aware of a published work that utilize embeddings in a sentiment (or classification) task specifically for Arabic text. However, research on Arabic sentiment analysis is getting more
attention from the research community recently. The majority of the existing work rely on manually engineered features in their classification. One of the most prominent features is the existence of certain words in a sentiment lexicon. Lexical-based features have been heavily exploited in almost all reviewed work of Arabic sentiment analysis [4, 5, 75, 6, 25].

In [4], they used both morphology-based and lexical features for subjectivity and sentiment classification of Arabic. Where POS tagging and stemming features were the key strength for [75] to achieves a remarkable performance on the subjectivity of Standard Arabic. As well as a whole corpus of labeled emotion words of Standard Arabic was presented in [3] for subjectivity and sentiment analysis.

On the other hand, a set of open issues and difficulties in Arabic sentiment analysis has been surveyed in [34].

4.1.2 Experiment design, implementation, and validation

Word Embeddings in Arabic Language

Semantics representation is a challenging task in natural language processing. Nonetheless, with the recent advancement in neural word representations models [68, 69, 58, 84], word embedding\(^{32}\) has emerged as the main spectrum for distributional semantic models. For the first time, distributed representations of words make it possible to capture words semantics; even the shift in meaning of words over time [66]. Such capability explains the recent successful switch in the NLP field from linear models over sparse inputs\(^{33}\), e.g. support vectors machines and logistic regression, to non-linear neural-network models over dense inputs [40]. Consequently, systems that rely on word embedding have been very successful in recent years, across a variety of NLP tasks [64].

Neural word embeddings are prediction-based models. In other words, in order to come up with distributed representations for words, the network learns its parameters by predicting the correct word (or its context) in a text window over the training corpus.

While the point of network training is to learn good parameters, word vector representations follow the notion that similar words are closer together\(^{34}\). In linguistics, this is known as “Distributional Hypothesis”\(^{35}\). This underlying idea is beneficial for extracting features from text represented in

\(^{32}\)Originally introduced in 2003 [19].

\(^{33}\)Although, a very recent work shows “re-discovers” the effectiveness of linear models through proposing a simple and efficient linear model for text classification and representation learning [51].

\(^{34}\)“You shall know a word by the company it keeps.” (Firth, J. R. 1957).

\(^{35}\)“Deep Learning for NLP” by R. Socher, 2016.
Table 6: Corpus collections and sources

<table>
<thead>
<tr>
<th>source</th>
<th>word count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quran-text</td>
<td>751,291</td>
</tr>
<tr>
<td>watan-2004</td>
<td>≈ 106 million</td>
</tr>
<tr>
<td>cnn-arabic</td>
<td>≈ 24 million</td>
</tr>
<tr>
<td>bbc-arabic</td>
<td>≈ 20 million</td>
</tr>
<tr>
<td>consumer reviews</td>
<td>≈ 40 million</td>
</tr>
</tbody>
</table>

such way; especially for understanding the context of word use in sentiment analysis. Since this notion is viable for any natural language, we take advantage of that and apply it to Arabic. Next we describe our approach for collecting an Arabic corpus and generating word vectors from the corpus.

**Building and Preprocessing Corpus**

We build a corpus from a set of publicly available text collections. Text contents are mainly news articles based on a local Arabic newspaper [2] and Arabic editions of international news networks [36] [88]. To enrich the corpus with dialectal vocabulary, we also include a pool of around 63 thousand consumer reviews [12], which include a mixture of different spoken Arabic. Further, as observed by [75], some social media users may tend to convey their feelings through using some verses from the holy Quran. Therefore, we include the complete text of Quran [37] in our corpus as to account for such sentiments. The complete corpus contains around 190 million words. Table 6 shows the corpus details with sources.

Next, we need to ensure the formatting of the sentences and words before we generate the embedding. Although most of the text collections are prepared properly since they come from published work, we notice few irregularities in some chunks of the text such as non-arabic letters and mis-placed punctuations. So we perform further preprocessing (e.g. extracting a sentence tokens then re-joining them) on the complete merged text. Python’s NLTK [38] was our perfect assist for sequencing the text. Though we ran into some encoding issues [39] that were eventually cleared.

---

36Such as CNN and BBC.
37Quran text source: http://tanzil.net (we used the simple-text version).
38http://www.nltk.org/api/nltk.tokenize
39Due to the difference between Python2 and Python3 in handling character unicode.
4.1 Building Embeddings From Raw Text

Generating Word Vectors

There are several models available for learning word embeddings from raw text. Among these are GloVe [84] and dependency-based word embeddings [60]. Our choice, however, is the well known and widely used word2vec model [68, 69]. Word2vec describes two architectures for computing continuous vectors representations, the skip-gram and Continuous Bag-Of-Words (CBOW). The former predicts the context-words from a given source word, while the latter does the inverse and predicts a word given its context window. We use CBOW to learn the embeddings; since it is simpler, computationally-efficient, and suitable for larger datasets [17].

Embeddings experiment setup

We are not sure about a good choice of hyper-parameters, so we ran several experiments with different settings. With each run, we would apply the generated embeddings to the sentiment classifiers (a sort of brute-force) to test performance. We tried arbitrary choices for window size (5, 10, 15, and 20) and embedding dimensions (200, 300, 500, 700). Based on the classifiers’ performance, window size 10 with 300 dimensions seems to be our best choice for learning good embeddings. Since the average size of each tweet in our dataset is 8.5 words, a window size of 10 would make sense.

From the available implementations, we experimented on both the original C implementation of word2vec toolkit [41] and Python’s gensim [42]. Training time takes around 102 minutes on a machine with 3.1GHz CPU and 16GB of RAM. From around 190 million words in our training corpus, the learned embedding size is 159,175 vocabulary.

Evaluating the embeddings

For English embeddings, vectors quality can be evaluated using a test set that comprises around 20,000 semantic and syntactic questions. However, for Arabic, we are not aware of any evaluation method. The actual quality measure, as we mentioned earlier, would be the classifiers’ performance down the road. We create multiple similarity and analogy queries to see how reasonable our embeddings are. Table ?? shows examples of the performed semantic analogy queries; while in Table ??, we show examples of sentiment-related results of the embeddings.

---

40 A modified version of word2vec
41 https://code.google.com/archive/p/word2vec/
42 https://radimrehurek.com/gensim/models/word2vec (Note: gensim library is a bit more sensitive to Arabic encoding, and it requires sequencing the input text.)
Dealing with dialectal Arabic is difficult. One of the problems is the use of different spellings for the same word. For example, some users may violate spelling rules of a certain word by adding or omitting letters. Fortunately, word embeddings work well in this situation. Since the different (misspelled) word variations are mostly used in the same context, they will have similar word vectors. Thus, we know they are semantically the same even though we actually did not specifically handle these violations. In our approach, we take this issue into account; that is why we include dialect text (the consumer reviews) in the training corpus. This will save a lot of efforts spent during preprocessing to handle such variations. Table ?? shows similarity results of two words in dialectal Arabic. We include the transliteration\textsuperscript{43} in the table to show how each word is written differently.

\textsuperscript{43}Transliteration is based on Buckwalter. See: http://www.qamus.org/transliteration.htm
Sentiment and Subjectivity Analysis

Datasets

We use three different datasets in our classification experiments. For the sentiment classification, we use twitter and book reviews datasets; we assume these two as “dialectal Arabic” text. Whereas for subjectivity classification we use news articles dataset, which is “Standard Arabic” text. The book reviews is scraped by [12] from a community driven website. While for news articles dataset, we use the labeled news articles of [16] which is based on an automatically translated Arabic version of MPQA corpus.

Table 7: Our collection of Twitter datasets and the source of each.

<table>
<thead>
<tr>
<th>Dataset name &amp; source</th>
<th>positive</th>
<th>negative</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTD (Nabil et al., 2015)</td>
<td>777</td>
<td>812</td>
<td>1589</td>
</tr>
<tr>
<td>ArTwitter (Abdulla et al., 2013)</td>
<td>993</td>
<td>958</td>
<td>1951</td>
</tr>
<tr>
<td>QCRI (Mourad and Darwish, 2013)</td>
<td>377</td>
<td>377</td>
<td>754</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>2147</strong></td>
<td><strong>2147</strong></td>
<td><strong>4294</strong></td>
</tr>
</tbody>
</table>

For twitter dataset, we were not able to acquire a sufficient number of labeled tweets. So, we rely on previously published datasets in [76, 6, 75] to compile a relatively larger set of tweets. To easily distinguish these datasets, we refer to them as ASTD, ArTwitter, and QCRI respectively. Originally, ASTD tweets were grouped in four categories (positive, negative, neutral, and objective); however, we sampled only from the positive and negative tweets. Where QCRI set comprised seven categories, again we only consider the positive and negative tweets. For better training, we balanced the selected samples. Table 7 shows number of samples we used from each dataset. After being combined together, we preprocess all tweets to remove non-arabic letters (e.g. handle names and mentions) and special characters (e.g. urls), we preserve the hashtags and emoticons however. The average number of tokens (words) in each tweet is 8.5.

---

44http://www.goodreads.com
4.1 Building Embeddings From Raw Text

(a) Sentiment of ASTD-ArabicTwitter-QRCI dataset

(b) Sentiment of LABR book review dataset

(c) Subjectivity of MPQA Arabic dataset

Figure 20: Learning curves of NaïveBayes and SupportVectors on Sentiment and Subjectivity datasets.
4.1 Building Embeddings From Raw Text

Training Classifiers

We consider the sentiment task as a standard binary classification. Thus, we run our experiments on six different binary classifiers with the three datasets. Table 9 include the full list of the classifiers with the performance of each. We use the implementations of [27] in all our classification experiments. We run all classifiers under same training conditions and we perform a 10-fold cross-validation with data split 90% train and 10% test. Classifiers performance will eventually determine the quality of our word representations. Although we do not attempt to tune classifiers for better results, SVM classifier and logistic regression classifier performed better than others in most of baseline results.

Features representation

In one way or another, methods that use manually-extracted features rely on sparse representations; in which each feature (e.g. words or POS tags) is its own unique dimension. In contrary to that, we represent each feature as a fixed-size dense vector. For each input sample, we obtain its feature vector by averaging the retrieved embeddings of that sample. Depending on the quality of the embeddings, similar features will end up having similar vectors. And here where the “powerful” key idea of the dense representations comes, generalization [40]. This enables our model to map (represent) unseen samples to similar feature vectors of those in the training set.

Table 8: Methods performance compared on the MPQA subjectivity of Standard Arabic.

<table>
<thead>
<tr>
<th>Method</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F-Score</th>
<th>MAcc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banea et al. [16]</td>
<td>72.30%</td>
<td>71.13%</td>
<td>71.30%</td>
<td>72.22%</td>
</tr>
<tr>
<td>Mourad et al. [75]</td>
<td>78.10%</td>
<td><strong>75.70%</strong></td>
<td>76.05%</td>
<td>77.20%</td>
</tr>
<tr>
<td>Our word embedding method</td>
<td><strong>79.95%</strong></td>
<td>72.67%</td>
<td><strong>76.14%</strong></td>
<td><strong>77.87%</strong></td>
</tr>
</tbody>
</table>

Results

Since our main focus is on the quality of the embeddings, we do not attempt to tune the classifiers parameters. Hence, we run classifiers with the simplest configurations possible (i.e. default parameters). Figure 20 shows a comparison between the training of Naïve Bayes classifier and SVM
classifier over the three datasets. From the learning curves, we can clearly notice that NB classifier will not benefit from adding more training data; since it converges quickly even with the small twitter dataset. Whereas in the case of SVM classifier, as we see in fig. 20 (a), there is a room to further improve our results by increasing the size of twitter dataset.

The detailed performance of all classifiers on each of the three datasets are reported in Table 9 and figure 21. For the purpose of method comparison, we follow the conventions in reporting our classifiers results. We use the measures recall, precision, F-measure, and macro-accuracy which correspond to Rec., Prec., F1, and MAcc respectively.

**Comparison with other methods**

For a fair comparison, we use the same dataset and task reported in the papers that we compare our work with. Table 8 shows our results on the subjectivity classification task compared to top reported work [16, 75] on the Standard Arabic dataset MPQA. Both [16] and [75] use hand-crafted features to achieve their results. The improvement in the latter method is attributed to POS tagging and word stemming they use. As we see in the table, our method achieves a slightly better accuracy than both of the other two. Though, we think, we still could improve upon our result should we use a larger training dataset, as we see from the learning curve of SVM in fig. 20 (c).

**Conclusion**

We have introduced neural word embeddings as an alternative of hand-crafted features for Arabic sentiment analysis. We exploit state-of-the-art word representations method to compute continuous vector representations of Arabic words. For the purpose of this work, we have built a large Arabic corpus to generate word representations. In our classification experiments, we relied solely on the dense representations of our embedding as the source of features; and yet we achieve a significant performance in compare to existing techniques. Our results showed that such a simple yet powerful method is enough to achieve state of the art performance. This should encourage research in the application of Arabic sentiment analysis to adapt more future-promising techniques. To further contribute to research in this area, we release our code and data used in our experiments.

In future work, we hope to compare the impact of different embedding methods (e.g. GloVe and skip-gram) on the performance of the classifiers. We also hope to see some clear and specific techniques

---

Footnote: For our comparison, we were able to get MPQA dataset only. Unfortunately, we could not obtain the other dataset “ArabSenti”; although we have contacted the original authors.
4.1 Building Embeddings From Raw Text

(a) ASTD-ArTwitter tweets

(b) LABR book reviews

(c) MPQA Arabic articles

Figure 21: Classifiers ROC on each dataset.
Table 9: Classifiers and their scores on each dataset. (Note: Rec., Prec., and MAcc. scores are the average of both positive and negative classes).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Measure</th>
<th>Classifier</th>
<th>LinearSVC</th>
<th>Rnd.Forest</th>
<th>GaussianNB</th>
<th>NuSVC</th>
<th>Log.Reg.</th>
<th>SGDClassifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astd-Artitter-QRCI (Sentiment)</td>
<td>Rec.</td>
<td>74.19%</td>
<td>71.43%</td>
<td>58.53%</td>
<td>76.50%</td>
<td>77.42%</td>
<td>75.58%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prec.</td>
<td>82.14%</td>
<td>75.98%</td>
<td>76.97%</td>
<td>83.00%</td>
<td>81.16%</td>
<td>82.00%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F1</td>
<td>77.97%</td>
<td>73.63%</td>
<td>66.49%</td>
<td>79.62%</td>
<td>79.25%</td>
<td>78.66%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAcc.</td>
<td>78.80%</td>
<td>74.17%</td>
<td>69.86%</td>
<td>80.21%</td>
<td>80.21%</td>
<td>79.53%</td>
<td></td>
</tr>
<tr>
<td>LABR-book-reviews (Sentiment)</td>
<td>Rec.</td>
<td>81.48%</td>
<td>80.23%</td>
<td>50.06%</td>
<td>81.73%</td>
<td>82.60%</td>
<td>88.74%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prec.</td>
<td>80.27%</td>
<td>79.04%</td>
<td>71.17%</td>
<td>80.82%</td>
<td>80.50%</td>
<td>73.17%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F1</td>
<td>80.87%</td>
<td>79.63%</td>
<td>58.78%</td>
<td>81.27%</td>
<td>81.58%</td>
<td>80.20%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAcc.</td>
<td>81.27%</td>
<td>80.05%</td>
<td>64.85%</td>
<td>81.69%</td>
<td>81.88%</td>
<td>78.60%</td>
<td></td>
</tr>
<tr>
<td>MPQA-Arabic (Subjectivity)</td>
<td>Rec.</td>
<td>72.67%</td>
<td>70.28%</td>
<td>58.57%</td>
<td>72.02%</td>
<td>77.87%</td>
<td>81.13%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prec.</td>
<td>79.95%</td>
<td>77.70%</td>
<td>75.42%</td>
<td>75.42%</td>
<td>78.30%</td>
<td>71.80%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F1</td>
<td>76.14%</td>
<td>73.80%</td>
<td>65.93%</td>
<td>75.03%</td>
<td>74.71%</td>
<td>75.40%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAcc.</td>
<td>77.87%</td>
<td>76.65%</td>
<td>71.16%</td>
<td>77.60%</td>
<td>75.60%</td>
<td>75.60%</td>
<td></td>
</tr>
</tbody>
</table>

for evaluating the quality of the embeddings.
4.2 Context-Based Approach For Learning Knowledge Graph Embeddings

You shall know a relation by the entities it connects.

In this chapter we develop an approach for building a representation model for Knowledge Graphs.

**Approach and objective**

Use neural networks to build a representation model for a Knowledge Graph. Such representation is known as Knowledge Graph Embeddings. The objective is to encode the components (i.e. entities and relations) of a knowledge graph into continuous vector spaces, so as to simplify the manipulation while preserving the inherent structure of the KG.

To easily grasp the proposed approach, this section is divided into four subsections. We start by laying out the general configuration of the proposed approach in section 4.2.1; then in section 4.2.3 we dive into a detailed description of the proposed method and its algorithm. And finally we discuss the used datasets and the validation process in sections 4.2.2 and 4.2.4 respectively.

**4.2.1 Data sources and structure**

Our approach uses a shallow neural network for learning distributed representations for knowledge graphs. Hence, we will need some training data to train our model. For this purpose, the training datasets can be either unstructured natural language sentences (where we infer relations between entities), or structured relational databases (where relations and entities are explicitly identified).

In this dissertation work we use the latter, which is more common for learning relational representations. A structured database is called knowledge base when it is intended to convey commonsense knowledge (knowledge about everyday life) or expert knowledge (domain specific). Knowledge Bases (KBs) store factual information as binary relationships between entities in the form of triplets. KBs range from general (e.g. Freebase, WordNet, Wikibase, YAGO, NELL) to more specialized (e.g. GeneOntology).

In mathematics, a binary relation is a set of ordered pairs of objects. Pairs that are part of the set are said to have a relation, or no relation otherwise. In the context of AI, a relation is considered to be a sentence in a syntactically simple and highly structured language. In which the relation takes the role of a verb, while two arguments are its subject and object. These sentences are expressed
as a triplet of tokens: \((\text{subject}, \text{verb}, \text{object})\)\(^{47}\), with the values: \((\text{entity}_i, \text{relation}_k, \text{entity}_j)\).

For example, to express that an engine is part of a vehicle, we could define the relation \textit{partOf} with the two entities \textit{engine} and \textit{vehicle} as the triplet \((\text{engine}, \text{partOf}, \text{car})\).

Let \(\mathcal{E}\) be the set of all entities in a domain and \(\mathcal{R}\) be the set of all relation types, a binary relation given by \(\mathcal{R}_r \subseteq \mathcal{E} \times \mathcal{E}\) is the subset of all pairs of entities for which the relationship is true.

Link prediction is a prominent problem in knowledge bases, where the goal is to predict missing, or incomplete, links (see figure 3). Learning efficient embeddings for triplets can be exploited for better KB completion. In the following section, we dive into detailed description of the proposed approach for learning such embeddings.

### 4.2.2 Data collection and description

Our experiments are based on two benchmark datasets WordNet \cite{wordnet} and Freebase \cite{freebase}. \textbf{WN18}: WordNet is a large lexical database of English that groups words into synonyms (or synsets) and provides lexical relationships between words. It is designed and maintained manually by professional linguists. \textit{WN18} datasets is a subset of WordNet. Table \ref{tab:stats} show some example triplets from this dataset. \textbf{FB15k}: Freebase is a large knowledge base with general facts about the world; it contains around 1.2 billion triplets and more than 80 million entities. It is harvested automatically from web sources such as wikipedia, and MusicBrainz.

Both of WN18 and FB15k are subsets from their original larger datasets. They were preprocessed and prepared specifically for relation embeddings task by \cite{dettmers2018convolutional}. Further, they are used heavily in many similar work. This will make it easier to evaluate how well our model perform compared to other models. See table \ref{tab:stats} for the statistics of these two benchmark datasets.

#### Table 10: Statistics of the experimental datasets we use.

<table>
<thead>
<tr>
<th>DATASET</th>
<th>entities</th>
<th>relationships</th>
<th>total triplets (facts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WN18</td>
<td>40,943</td>
<td>18</td>
<td>151,442</td>
</tr>
<tr>
<td>FB15k</td>
<td>14,951</td>
<td>1,345</td>
<td>592,213</td>
</tr>
</tbody>
</table>

#### Table 11: Relation type frequencies in WN18

<table>
<thead>
<tr>
<th>relation</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>_hyponym</td>
<td>34832</td>
</tr>
<tr>
<td>_hypernym</td>
<td>34796</td>
</tr>
<tr>
<td>_derivationally_related_form</td>
<td>29715</td>
</tr>
</tbody>
</table>

\(^{47}\)Some sources use the RDF standard for representing facts (subject, predicate, object)
4.2 Context-Based Approach For Learning Knowledge Graph Embeddings

<table>
<thead>
<tr>
<th>relation</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>_member_meronym</td>
<td>7402</td>
</tr>
<tr>
<td>_member_holonym</td>
<td>7382</td>
</tr>
<tr>
<td>_has_part</td>
<td>4816</td>
</tr>
<tr>
<td>_part_of</td>
<td>4805</td>
</tr>
<tr>
<td>_member_of_domain_topic</td>
<td>3118</td>
</tr>
<tr>
<td>_synset_domain_topic_of</td>
<td>3116</td>
</tr>
<tr>
<td>_instance_hypernym</td>
<td>2935</td>
</tr>
<tr>
<td>_instance_hyponym</td>
<td>2921</td>
</tr>
<tr>
<td>_also_see</td>
<td>1299</td>
</tr>
<tr>
<td>_verb_group</td>
<td>1138</td>
</tr>
<tr>
<td>_member_of_domain_region</td>
<td>923</td>
</tr>
<tr>
<td>_synset_domain_region_of</td>
<td>903</td>
</tr>
<tr>
<td>_synset_domain_usage_of</td>
<td>632</td>
</tr>
<tr>
<td>_member_of_domain_usage</td>
<td>629</td>
</tr>
<tr>
<td>_similar_to</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 12: Sample KB triplets for “molecule” entity in WN18

<table>
<thead>
<tr>
<th>head</th>
<th>relation</th>
<th>tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>__radical_NN_1</td>
<td>_part_of</td>
<td>__molecule_NN_1</td>
</tr>
<tr>
<td>__physics_NN_1</td>
<td>_member_of_domain_topic</td>
<td>__molecule_NN_1</td>
</tr>
<tr>
<td>__molecule_NN_1</td>
<td>_has_part</td>
<td>__atom_NN_1</td>
</tr>
<tr>
<td>_unit_NN_5</td>
<td>_hyponym</td>
<td>__molecule_NN_1</td>
</tr>
<tr>
<td>__chemical_chain_NN_1</td>
<td>_part_of</td>
<td>__molecule_NN_1</td>
</tr>
<tr>
<td>__chemistry_NN_1</td>
<td>_hypernym</td>
<td>__unit_NN_5</td>
</tr>
<tr>
<td>__molecule_NN_1</td>
<td>_member_of_domain_topic</td>
<td>__molecule_NN_1</td>
</tr>
<tr>
<td>__molecule_NN_1</td>
<td>_synset_domain_topic_of</td>
<td>__physics_NN_1</td>
</tr>
<tr>
<td>__molecule_NN_1</td>
<td>_has_part</td>
<td>__radical_NN_1</td>
</tr>
<tr>
<td>__molecule_NN_1</td>
<td>_hyponym</td>
<td>__supermolecule_NN_1</td>
</tr>
<tr>
<td>__atom_NN_1</td>
<td>_part_of</td>
<td>__molecule_NN_1</td>
</tr>
</tbody>
</table>

4.2.3 Hybrid context-based training algorithm for relational data

Based on neural networks approach for statistical language modeling, the proposed method is inspired by the Continuous Bag-Of-Words model for training word vectors (see word embedding in section 2.2). The core difference is that in our case we treat each triplet as an individual word.

Provided a knowledge base “as the training data”, we learn distributed representations for entities and relations by considering each triplet as training example; and then maximizing a training objective that capture their joint distribution.

- The basic idea is to apply the Distributional Similarity Hypothesis on relational data (knowledge bases).
- We treat each triplet in the training set as a target training example (just like how a target word is treated in CBOW model). Its context is formed as follows:
4.2 Context-Based Approach For Learning Knowledge Graph Embeddings

- The context will be other training triplets from the training set that has the same relation type as the target triplet. Since, we adopt negative sampling method for the training, a context is sampled such that it contains true and negative training examples. Such that:
  - true samples: correct triplets from the training data where each triplet has the same relation type but different entities.
  - negative samples: corrupted (not true) triplets with same relation type, but different entities drawn randomly from the set of training entities $\mathcal{E}$.

For example, given the training examples in table 13, we have the set of entities $\mathcal{E} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, and the set of relations $\mathcal{R} = \{r_1, r_2, r_3\}$. Assuming the context “window” size is two and the current training target is the triplet $(e_1, r_2, e_3)$, then its context would be the triplets $(e_6, r_2, e_4)$ and $(e_2, r_2, e_5)$.

Table 13: An example training dataset.

<table>
<thead>
<tr>
<th>training examples - triplets</th>
</tr>
</thead>
<tbody>
<tr>
<td>(e3, r1, e2)</td>
</tr>
<tr>
<td>(e1, r2, e3)</td>
</tr>
<tr>
<td>(e1, r3, e4)</td>
</tr>
<tr>
<td>(e2, r2, e5)</td>
</tr>
<tr>
<td>(e6, r2, e4)</td>
</tr>
<tr>
<td>(e3, r1, e6)</td>
</tr>
<tr>
<td>(e5, r3, e3)</td>
</tr>
</tbody>
</table>

With these conventions, our objective is to maximize the log-likelihood of each triplet given its context:

$$
\frac{1}{|\mathcal{R}|} \sum_{r=1}^{R} \sum_{j \in C} \log p(t_i^r | t_j^r)
$$

(6)

where, $\mathcal{R}$ is the total relation types in the training examples. $C$ is the context of the triplet $t_i$, such that the relation type in $t_i$ is the same relation type in $t_j$. And $t$ is a compositional vector representation of the triplet, given by:

$$
t^r = d(e_s, r_v, e_o) = ||e_s + e_o - r_v||_{1/2}
$$

(7)

where, $d$ is the dissimilarity measure we use in the score function\textsuperscript{48}. And $e_s$ is the subject entity and $e_o$ is the object entity $\in \mathcal{E}$ and $r_v$: relation type $\in \mathcal{R}$.

\textsuperscript{48}we use a similar distance function as the one described in [24].
The probability of a relation given its context triplets \( p(t_i^r | t_j^r) \) can be calculated using \texttt{softmax} as follows:

\[
p(t_i^r | t_j^r) = \frac{\exp(t_{jr} \cdot t_{ir})}{\sum_{t^r_k \in \mathcal{R}} \exp(t_{rk} \cdot t_{jr})}
\]

(8)

For optimizing our objective, i.e. learn vector representations of the entities and relationships jointly, we use the training algorithm 8 along with the well established Stochastic Gradient Descent “SGD” to update the model’s parameters. The proposed algorithm is a hybrid of two published work. That is our training method is inspired by CBOW of [68], while the score function and its optimization is adopted from [24].

\textbf{Algorithm 8} Our Contextual Knowledge Graph Embedding algorithm for semantic relations into a low-dimensional vector space

\begin{itemize}
  \item \textbf{Input} : Triplet set \( T = \{(h, r, t)\} \) i.e. entities \((h, t) \in \mathcal{E} \) and relation \( r \in \mathcal{R} \).
  \item \textbf{Params}: dimension \( d \), learning \( \epsilon \), window \( w \), epoch \( n \), and batch size \( b \).
  \item \textbf{Output} : Embedding matrices \( E \in \mathbb{R}^{||\mathcal{E}|| \times d} \) and \( R \in \mathbb{R}^{||\mathcal{R}|| \times d} \).
  \item initialize \( r \leftarrow \text{uniform} (-\frac{6}{\sqrt{d}}, \frac{6}{\sqrt{d}}) \) for \( r \in \mathcal{R} \).
  \item initialize \( e \leftarrow \text{uniform} (-\frac{6}{\sqrt{d}}, \frac{6}{\sqrt{d}}) \) for \( e \in \mathcal{E} \).
  \item \textbf{while} epochs do
    \quad \text{ \( T_{batch} \leftarrow \text{sample} \ (T, b) \)} \quad \text{// training set}
    \quad \text{\( S \leftarrow \emptyset \)}
    \quad \text{\textbf{for} \( t \) \textbf{in} \( T_{batch} \) \textbf{do}}
      \quad \text{context will return \( w \) true triplets with same relation type \( r \)}
      \quad \text{\( C \leftarrow \text{context} \ (t, T_{batch}, w) \)} \quad \text{// \( t \) is a triplet \((e1, r, e2)\)}
      \quad \text{\( S \leftarrow S \cup \{(t, C)\} \)}
    \quad \text{\textbf{end}}
    \quad \text{update weights using SGD w.r.t \( \epsilon \) and loss function} \sum_{t \in S} \nabla \log p(t | t_{[C]})
  \quad \textbf{end}
\end{itemize}

As the computation of \( \nabla \log p(t_i^r | t_j^r) \) is proportional to \( \mathcal{R} \); the cost of computing the normalization part for every training example (i.e. the denominator part of softmax function, see equation 8) can be too expensive. So in practice, an alternative such hierarchical softmax or negative sampling method can be used instead of softmax, as described in [71]. Also in a very recent work, in [44] they have proposed “Adaptive Softmax” as an efficient approximation based training method which cut the linear dependency on the vocabulary size.

4.2.4 Validation process

To evaluate the performance of our approach, we follow the ranking evaluation protocol described in [24]. For each test triplet, the head entity will be removed and replaced by each of the entities
from \( \mathcal{E} \). The model will then computed the predictions of these corrupted triplets and sort them in ascending order. The same procedure is repeated again but with removing the tail entity. Then take the average of these is prediction ranks. The common metrics used for this process includes hits@1, hits@3, and hits@10.

This evaluation procedure is used to evaluate link prediction tasks; where it measures the portion of correctly predicted triplets from a test set (i.e. the portion of test triplets in which the target entity was predicted correctly). The same evaluation protocol is adopted in literature in many related work [109, 82, 78, 79] which makes comparing the accuracy of our results viable. To facilitate a fair comparison, we are using the same datasets used in the mentioned papers.

To measure the quality of the relations ranking, we will also report the mean reciprocal rank (MMR):

\[
\text{MMR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \quad (9)
\]

which is a commonly used measurement metric in information retrieval tasks.
5 Experimental Evaluation and Validation

In this chapter, we introduce our experimental tests where we 1) did extrinsic and intrinsic evaluations to determine the accuracy of embeddings models, and 2) applied query answering system based on knowledge graphs to assess the performance of knowledge graph embeddings.

The following section is based on our published journal paper [11].

5.1 Evaluating the Quality of the Embedding Vectors

Abstract. We consider the following problem: given neural language models “embeddings” each of which trained on an unknown data set, how can we determine which model would provide a better result when used for feature representation in a downstream task such as text classification or entity recognition?

In this paper, we assess word similarity measure through analyzing its impact on word embeddings learned from various datasets and how they perform in a simple classification task. Word representations were learned and assessed under the same conditions. For training word vectors, we use the implementation of Continuous Bag Of Words described in [68]. And to assess the quality of the vectors, we apply the analogy questions test for word similarity described in the same paper. Further, we introduce a new metric to measure the retrieval rate of an embedding model. It measures the percentage of missing words in the model. We call it Average Retrieval Error.

We observe that scoring a high accuracy of syntactic and semantic similarities between word pairs is not an indicator of better classification results. And that can be justified by the fact that a domain-specific corpus contributes to the performance better than a general-purpose corpus. We also discover that word retrieval average of the matched vocabulary in a model does not affect the overall performance as well. We think one way to enrich this metric is by introducing word-wise weights. For reproducibility, we release the scripts and results of our experiment[49].

Introduction

Language modeling is the crux of the problem in Natural Language Processing. Recently, neural language models have outperformed the traditional language model approaches such as n-gram. The

[49]https://github.com/iamaziz/embed-eval
superiority of the neural methods lies in their capability to overcome the curse of dimensionality problem while, simultaneously, capturing different similarities between words [43].

Neural language models learn distributed representation for each word in the form of real-numbers-value vectors, which allows similar words to have similar vectors. Such sharing is an important characteristic that enables the learning model to treat related words similarly and, hence, the ability to generalize. These word representations are usually known simply as Word Embeddings.

Nowadays, word embedding is the standard approach for feature representation in many NLP tasks. Traditional feature representation methods, such as bag-of-words and its TFIDF, rely on hand-crafted feature extractor, time-consuming, and domain-specific. Hence, embedding based techniques provide a better alternative for automating many tasks in language modeling and NLP.

Among these techniques, distinctly, context-predicting semantic vectors have proven their superiority to the count-based ones [18]. Context-based vectors make more emphasis on the word (and its context) and other words and contexts.

Popular word vector learning methods are introduced in [68, 84, 66] and have had gained great attention since then. From these methods, learning continuous word embeddings using skip-gram and negative sampling is the most common approach for building word vectors [63]. This method was described and introduced in [68].

However, since vectors training occurs in an unsupervised fashion; there is no accurate way to estimate the quality of the vector representations objectively. Several extrinsic and intrinsic evaluation methods have been discussed [15]. However, until the date of this writing, there is still no reliable method for comparing the quality of different embedding models. So, this is still an open question. Commonly, the quality can be assessed using the word similarity task, which is a test with a set of similarity analogy questions [68].

Nevertheless, using the current word similarity evaluation method, 1) word similarity accuracy, and 2) having more vocabulary in the model does not result in better performance in the downstream task.

From experiments, we show that scoring high accuracy in word similarity measure questions does not imply better performance in the downstream task. Our finding is in line with the observations of [35]. Therefore, we observe that the accuracy of word similarity measure is not, necessarily, an indicator for the usefulness of the word embedding model. In this paper, we explain and justify this claim based on the observation of our experimentation results.
5.1 Evaluating the Quality of the Embedding Vectors

For instance, we show that GoogleNews embedding model has the following two advantages over IMDB model. First, it scores better word similarity accuracy (74.26%) in comparison to IMDB’s (23.71%); second, GoogleNews contains 3 million vocabulary while IMDB contains around 19,000. Despite these advantages of GoogleNews, the classifiers’ performance was worse with GoogleNews than with IMDB.

The rest of the paper is structured into the following parts: related work, our experiments, discussion, future work, and finally the conclusion.

Related Work

We approached related work in the following manner. First, we investigated what it takes to build quality embedding models and which components to consider. We then analyzed similar work for evaluating word embeddings using extrinsic and intrinsic methods. We also reviewed the available current work on building domain-specific embeddings. And finally, we look into work that focuses on the syntactic and semantic similarities between words.

Training elements such as the model, the corpus, and the parameters have been analyzed in detail in [57]. They observed that the corpus domain is more important than its size. This explains our results where the smaller domain-specific corpus (IMDB) achieved better results than the much larger general-purpose corpus (GoogleNews).

We reviewed papers on evaluating word vectors’ quality and model accuracies. Existing evaluation methods fall into two types: intrinsic and extrinsic evaluation. In the intrinsic evaluation, the goal is to directly assess the quality of word vectors in the hope that it will reflect on the performance of downstream tasks. So, synthetic metrics are proposed to test the semantic and syntactic similarities between words.

For example, a pre-selected set of query terms is used to estimate words relationships. Each query denotes two pairs of “analogically” similar words (e.g. big to bigger as small to smaller “syntactic similarity”, or Tokyo to Japan as London to England “semantic similarity”). Then, such queries can take the form of questions e.g. “What is the word similar to small in the same sense as bigger is similar to big?”. To query the model, a question is formulated in an algebraic expression as follows: \( \text{answer} = \text{vector('bigger')} - \text{vector('big')} + \text{vector('small')} \). This method was first proposed in [68]; and published with a set of around 20 thousand syntactic/semantic questions. It is fast and computationally inexpensive, however, there are problems associated with this technique [35].
5.1 Evaluating the Quality of the Embedding Vectors

Further, other evaluation techniques have been proposed to reduce bias [91]. In such methods, they directly compare embeddings with respect to specific queries.

While in the extrinsic evaluation, we indirectly evaluate word embeddings. In other words, we use the embeddings as input features to a downstream task and measure the performance metrics specified to that task [91]. For instance, when the task is text classification, we would use the embeddings to represent words in the text. In some approaches, they applied extrinsic evaluations to learn task-specific embeddings [98].

Finally, a thorough investigation and survey covering the current evaluation methods have been discussed in [15].

Building Word Embeddings

Data Collection and Exploration

In this section, we describe the data sources and texts we used for training the embedding models. We started with two well-known corpora. The first one is text8, a standard corpus used in NLP community which has around 100MB of cleaned English text of a Wikipedia dump from 2006, and the second one is the Large Movie Review Dataset (or IMDB). IMDB contains 100 thousand movie reviews prepared for sentiment classification problems. Later on, we will use this same dataset in our classification experiment; we are aware this may cause bias in the datasets, further discussion to follow later.

As a way to augment our data, we created a new hybrid corpus by concatenating the above two corpora; we call it text8-imdb. This allows us to compare the results of two models and their hybrid to see how they may affect one another. Later on, in the classification section, we will see that imdb achieved the best among the three. This is a bit surprising, because its average retrieval error (1.46) was higher than that of text8-imdb (0.99); though it still achieved better results.

For additional insights about the data, we explored each corpus for statistical information “metadata” such as number of the unique words, the total count of characters, and the total count of words. See Table 14 for more details on these metrics.

\[http://mattmahoney.net/dc/textdata\]
\[http://ai.stanford.edu/amaas/data/sentiment/\]
5.1 Evaluating the Quality of the Embedding Vectors

Table 14: Statistics of the training text (corpus).

<table>
<thead>
<tr>
<th>Corpus</th>
<th>char count</th>
<th>word count</th>
<th>unique words</th>
</tr>
</thead>
<tbody>
<tr>
<td>imdb</td>
<td>125,882,839</td>
<td>23,573,192</td>
<td>144,841</td>
</tr>
<tr>
<td>text8</td>
<td>100,000,000</td>
<td>17,005,207</td>
<td>253,854</td>
</tr>
</tbody>
</table>

We also wanted to get a better sense of the characters’ usage and their frequency in each corpus. Figure 22 illustrates some visualization of the usages. It shows the frequency of the 26 English letters usage in each of the three corpora.

![Figure 22: Letters frequencies as they appear in text8 and IMDB](image)

Model Training and Parameters

Following [68] approach for learning vector representations of words, we trained three models using three various corpora. In the first one, we merged the entire set of 100,000 movie reviews [65] into one big text file, we will refer to the vectors “model” generated from this text as imdb. And for the second model, as mentioned in the previous section, we used a 100 MB of cleaned Wikipedia English text known as text8, we will call the model from this corpus: text8. The third “hybrid” model is the combination of the two above files (as one big text file). We refer to this model as imdb-text8.

The fourth model, in our experiment, is GoogleNews. A pre-trained model published in [68].

With the exception of GoogleNews, all the models were trained using CBOW architecture with the same hyper-parameters. We used the original (C language) implementation of word2vec toolkit. After compiling and building the software locally, we use the following command to train the models:

```
$ ./word2vec -train $CORPUS \ -output $OUT \
```

[52] It was not clear to us which exact parameters were used. See: Mikolov’s response in word2vec-gmail-group

[53] https://github.com/tmikolov/word2vec
Exploring the Models

After we built the models, we decided to evaluate their response to the analogy question test sets. The table below displays the number of the learned “vectorized” vocabulary in each model. The table also shows the number of questions seen in the model, along with their average similarity accuracy. These results were obtained based on $./word2vec/compute-accuracy$ script in the same toolkit. For faster approximate evaluation, we used the recommended threshold of 30,000 to reduce vocabulary.

Table 15: Embedding vectors compared.

<table>
<thead>
<tr>
<th>Embeddings</th>
<th># vocab.</th>
<th>dim.</th>
<th># quest. seen</th>
<th>avg. sim. acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>imdb</td>
<td>53,195</td>
<td>300</td>
<td>10,505</td>
<td>33.41%</td>
</tr>
<tr>
<td>text8</td>
<td>71,291</td>
<td>300</td>
<td>12,268</td>
<td>53.60%</td>
</tr>
<tr>
<td>imdb-text8</td>
<td>94,158</td>
<td>300</td>
<td>12,448</td>
<td>59.89%</td>
</tr>
<tr>
<td>GoogleNews</td>
<td>3M</td>
<td>300</td>
<td>13,190</td>
<td>76.85%</td>
</tr>
</tbody>
</table>

Determining Models Accuracy

To conduct a fair comparison between models, we introduce the Average Retrieval Error “AVG_ERR” as a way to estimate the vectors’ availability in the given model. It is the total number of missed words (i.e. words that queried but not available in the embedding model) over the total words queried. See formula 10 below:

$$\frac{\sum_{i=1}^{n} Q(t_i - r_i)}{n}$$  \hspace{1cm} (10)

Where, $Q$ is a query to the model which returns the vectors for a set of given tokens (words), $n$ is the total number of the queries made, $t$ is the number of tokens in query $i$, and $r$ is the number of retrieved (found) vectors for query $i$.

For simplicity, we can rewrite 10 as:
5.1 Evaluating the Quality of the Embedding Vectors

\[
\text{Avg. Retrieval Error} = \frac{\sum_{i=1}^{n} m_i}{n}
\]  

(11)

And \( m \) is the number of missed (not found) vectors for query \( i \).

In Figure 23, we show the number and percentage of analogy questions seen in the model (with a threshold of 30K) for word similarity task.

![Figure 23: Embeddings results on the word analogy task (out of the total 19544 questions), fig. a. is the number of questions seen and fig. b. is the percentage of the questions seen.](image)

We also recorded the accuracy for each topic of the 14 question type categories. Instead of using a huge table with many numbers, we decided to illustrate the result in figure 26 to quickly grasp the topics’ results.

![Figure 24: Results on the topics accuracy from word analogy task](image)

Finally, in figure 25, we show the overall accuracy results for every model; such as the average score
5.1 Evaluating the Quality of the Embedding Vectors for all topics and density of topics’ results.

Despite the scored word similarity accuracy of the IMDB model, its classification result is quite impressive. We will see that in the next section; where the learned word representations reflect a great deal of the actual semantics.

**Applying Embedding Models for Binary Classification**

In this section we evaluate the performance of each embedding model through a downstream task. Our task is a simple binary classification for sentiment analysis problem.

**Supervised Training Dataset**

To train the sentiment classifiers, we used the popular benchmark IMDB-50K movie reviews dataset. It was introduced by [65], and available to download. The dataset, which was prepared specially for binary sentiment classification, contains 25K highly polar movie reviews for training and 25K for testing. The sentiment of reviews is balanced in both data sets, i.e. one half is positive, and the other half is negative.

Additionally, IMDB has another unlabeled dataset contains 50K reviews which we used in training our word2vec models. This dataset, however, was not used for training the binary classifiers.

**Representing Reviews**

After preprocessing the review text, the vector representation of each token “word” is then retrieved by querying the embedding model. If a token is not found in the embeddings’ vocabulary, its

---

representation will be ignored. That’s where the concept of Average Retrieval Error comes from. The more tokens missed, the higher the average error will be. When all the review’s tokens are processed, the review then will be represented as a fixed size feature vector by averaging the representations of all tokens.

Training Classifiers Results

We trained five simple binary classification algorithms Perceptron, Support Vector Machines, Stochastic Gradient Descent, Logistic Regression, and Random Forest. We used the built-in implementations of these algorithms provided by the scientific toolkit library “scikit-learn”\(^\text{55}\). As for parameters tuning, we applied the default parameters in scikit-learn.

To know the complete set of parameters for each classifier, one can refer to the log file we included with our project code.

In table, we show the performance of each classifier with each of the respective four embedding models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Vocabulary Size</th>
<th>AVG_ERR</th>
<th>Percept.</th>
<th>SVM</th>
<th>SGD</th>
<th>LogReg</th>
<th>RForest</th>
</tr>
</thead>
<tbody>
<tr>
<td>imdb</td>
<td>53,195</td>
<td>1.46</td>
<td>84.29%</td>
<td>89.20%</td>
<td>86.49%</td>
<td>89.19%</td>
<td>84.39%</td>
</tr>
<tr>
<td>text8</td>
<td>71,291</td>
<td>4.62</td>
<td>76.62%</td>
<td>81.17%</td>
<td>75.44%</td>
<td>81.22%</td>
<td>73.88%</td>
</tr>
<tr>
<td>imdb-text8</td>
<td>94,158</td>
<td>0.99</td>
<td>80.11%</td>
<td>89.12%</td>
<td>85.50%</td>
<td>89.08%</td>
<td>83.96%</td>
</tr>
<tr>
<td>GoogleNews</td>
<td>3,000,000</td>
<td>28.04</td>
<td>78.94%</td>
<td>86.14%</td>
<td>82.89%</td>
<td>86.08%</td>
<td>80.16%</td>
</tr>
</tbody>
</table>

See figure for a better visual comparison of the scores. We can see that the classifiers scored better with IMDB embedding model, despite that GoogleNews model has better accuracy in term of analogy query test. We can also notice that IMDB is still better than its hybrid model text8-imdb which intuitively should enrich the model’s representation capacity by adding more vocabulary (which can be verified by inspected the average retrieval error decrease from imdb to text8-imdb). Reducing AVG_ERR did not improve the classifiers; but on the contrary, combining text8 degrades imdb’s performance.

Avoiding bias in IMDB

The training and testing datasets are initially the same corpus that we use to generate imdb embeddings. Thus, and to make sure that our testing is not biased, we used another sentiment dataset (i.e. other than IMDB reviews) to test the performance of the classifier. The dataset contains 7086

\(^{55}\text{https://scikit-learn.org}^\)
5.1 Evaluating the Quality of the Embedding Vectors

Figure 26: Sentiment classifiers score with each embeddings. a) embedding models wise results, and b) classifiers wise results.

Labeled (positive/negative) training sentences and 33052 unlabeled sentences provided for prediction problems. We used the training data for testing our classifiers, as we were not able to acquire the actual labels of prediction set. As expected, the highest scores of the classifiers still achieved with imdb embeddings.

Summary

Finally, and to summarize and aggregate all results and scores together in one place, we took the average score of all classifiers achieved with each embedding model. These aggregates are displayed in table 17.

Table 17: Summary on the final results for embedding models’ accuracy and classification performance

<table>
<thead>
<tr>
<th>Embeddings</th>
<th>vocab. size</th>
<th>AVG. retrieval err.</th>
<th>AVG. similarity acc.</th>
<th>AVG. sentiment score</th>
</tr>
</thead>
<tbody>
<tr>
<td>imdb</td>
<td>53.195</td>
<td>1.46</td>
<td>33.41%</td>
<td>86.73%</td>
</tr>
<tr>
<td>text8</td>
<td>71.291</td>
<td>4.62</td>
<td>53.60%</td>
<td>77.74%</td>
</tr>
<tr>
<td>imdb-text8</td>
<td>94.158</td>
<td>0.99</td>
<td>59.89%</td>
<td>85.55%</td>
</tr>
<tr>
<td>GoogleNews</td>
<td>3M</td>
<td>28.04</td>
<td>76.85%</td>
<td>82.79%</td>
</tr>
</tbody>
</table>
5.1 Evaluating the Quality of the Embedding Vectors

Discussion

Model Accuracy and Classifiers Performance

Why IMDB word embedding model is better than GoogleNews embedding? Learning task-specific vectors through fine-tuning offers further gain in performance. See static vs. non-static representation (section 4.2 of CNN sentence classification [55]).

So, for example, you’d expect words like “amazing” and “awful” to be very far apart whereas in word2vec they'd probably be closer because they can appear in similar contexts [56].

In the accuracy evaluation, IMDB model scored 22.94% on the 8182 test cases found (out of the 19544 test cases); while the GoogleNews model scored 74.26% on the 7614 test cases found. Although the IMDB model scored less, the sentiment classifiers performed better with it in comparison to the other model.

Things to notice

Although we were not concerned with improving the overall performance of the classifiers, there are several things to consider that can improve the classifiers’ results.

For example, one can apply the ensemble approach, described in [67], that combine multiple baseline models rather than relying on a single model. Further improvement might be introduced by describing the review feature differently, instead of averaging the vectors [59].

Also, while training the vectors, careful choice and tuning of the hyper-parameters could bring much gain to the model accuracy [61]. Finally, one may consider words dependency instead of relying solely on linear contexts [45].

Missing data

When a given token (of a sentence) is not available in the embedding model, its vector value is ignored. However, it is counted toward the sentence length when we take the overall average. Can we do something else about this? e.g. 1) substitute (compute) its value as the average of other tokens in the same review, or 2) do not count it in review length, or 3) apply other known techniques for handling NaN values.

56see: cnn-text-classification-tf Denny Britz’s blog
Future Work

We can think of three possible ways to further extend this work. Firstly, expand the models range for broader comparison. For instance, one can integrate more (other) pre-trained models such as GloVe, ELMo, BERT to use in both experiments; embedding quality assessment, and binary classifiers. Secondly, and to enrich the procedure of classification comparison, one can try another approach to aggregating the sentence features (other than averaging vectors for sentence representations). Finally, in this work, we introduced the Average Retrieval Error “AVG_ERR”. We think this measure can be further improved by adding weights to words in the sentences. For example, stop words, and common vocabulary can have less weight than those that are more specific.

Conclusion

We discussed the problem of choosing between multiple word embedding models. To this end, we made the following contributions. We built and trained three different embeddings models based on published data sets. We, then, implemented two types of evaluation methods on the models. For the intrinsic evaluation, we applied the word similarity measure method; while we did the extrinsic evaluations through a binary classification problem. We presented the results of performance comparisons over four different embedding models. We also introduced a metric for measuring the model’s retrieval rate to the number of queries made. For reproducibility, we released the models, data, and scripts used in our experiments.

We have shown that scoring high accuracy in the Word Similarity Measure test does not imply better performance in the downstream task. In other words, if a model A achieves a higher score than model B in the analogy question test, this does not mean A will perform better than B in a downstream task. This finding is in line with observations from related work. We also observed that the model’s coverage of vocabulary (i.e. vocabulary size) is not as essential as containing a domain-specific dictionary.
5.2 Applying KGE for Answering Fact-based Questions

Question Answering (QA) is a popular task in natural language processing (NLP). It is a generic framework for answering questions about specific information, for example weather status, soccer statistics, or factual knowledge. In the early days of artificial intelligence, systems used two major paradigms of question answering: information-retrieval-based and knowledge-based [53].

This work relies on a method for representing a set of pre-defined knowledge facts (i.e. knowledge graphs) into vector space models (VSMs). VSM is an excellent way to represent text in a machine-interpretable manner.

The embeddings of knowledge graphs can be utilized in several search-related applications such as search engines, dialog systems (e.g. question-answering and chatbots), and social network applications. In this section, we build a factoid question-answering system that leverages knowledge graph embeddings.

Question answering can be “open-domain” or “closed-domain”, in our system we focus on a specific “closed-domain” factoid type of questions based on the prior knowledge included in the FreeBase knowledge base. In the following, we describe our Q/A system and its domain in details.

5.3 KGE QA System

In this section, we introduce our Factoid-question answering systems. We call it Factoid-Question Answering System Based on Knowledge Graph Embeddings; for short KGE QA.

5.3.1 Algorithm and Application Screenshot

KGE QA Algorithm

In this section, we introduce the pseudocode for the KGE QA algorithm in 9. Also, since we built a web app for this system, we include sample screenshots from the main user interface. The screenshots describes information about the domain knowledge, how to ask questions, and sample questions and answers.
Algorithm 9 KGE QA Algorithm for answering factoid-questions

1. read the input question text
2. preprocess and tokenize the input string
3. remove stopwords
4. join True sub_words
5. for each token
6. if the token is a True key (i.e. in either ENT/REL models):
7. label its type as is and
8. continue to the next token
9. generate a vector for the token
10. find its closest neighbors in our (ENT/REL) models:
11. get top n closest ENTS and top n closest RELs (cosine similarity)
12. if the similarity distance (of the closest neighbor) is not close enough (i.e. less than a threshold)
13. decide the closest neighbor by taking the max of (SequenceMatcher + CosineSimilarity) / 2
14. if the distance of the closest matched neighbor is still not close enough (i.e. use a threshold 0.2)
15. label the token’s type as type OTHER i.e. neither ENT nor REL
16. remove non ENT/REL tokens (i.e. ‘OTHER’) from the input tokens
17. swap the tokens’ WORDS from the “INPUT” words to “ENT/REL” words in our models
18. form incomplete triplet pairs (head-relation) from the labeled tokens
19. pick the first True pair
20. answer is the tail(s) of closest triplet(s) in our KG dataset
KGE QA System

Factoid-Question Answering System Based on Knowledge Graph Embeddings

Info

This system answers simple factual questions such as who are the characters of titanic movie? or what’s the spoken language in Indonesia? The goal is to demonstrate how to leverage Knowledge Graphs Embeddings to build such system.

What kind of questions can be asked?
This system supports a specific domain knowledge.
To make sure the asked question falls under the covered domain knowledge, a question should contain at least one entity (head) and one relation from the domain knowledge facts.
See 'Explore the domain knowledge' on the left sidebar to learn more about the supported facts and their ENTities/RELations.

NOTICE: A key characteristic of the system, however, is that it is possible to ask about the same piece of information in a variety of (syntactic/semantic) linguistic forms.
In other words, the same question might be asked in different forms e.g.
- Troy movie is written by whom?
- who wrote the movie troy?

Figure 28: Screenshot of the usage information

KGE QA System

Factoid-Question Answering System Based on Knowledge Graph Embeddings

Info

Enter your question:

**titanic movie directed by whom?**

Answer:

- 0
- 0 James

![Diagram of directed_by relationship between titanic and James]

Figure 29: Example1: uncleaned FB15K dataset: example question and answer
5.3 KGE QA System

**KGE QA System**

*Factoid-Question Answering System Based on Knowledge Graph Embeddings*

[Image: Input: who is the writer of Troy movie?]

**Figure 30:** Example 2: sample question and answer

[Diagram: Visualizing closest relations based on input question]

**Figure 31:** Example 2: visualizing the closest relations based on the input question
5.3.2 How we build ENT and REL models

Given a knowledge graph facts in triples format, our objective is to build two separate embedding models. One model is for entities and another model is for relations. We start by extracting the set of all entities (both heads and tails) in the triples as well as the set of relations.

The vectors are generated using a pre-trained embedding model. Literature contains various benchmark pre-trained models. We tested on few popular embedding models provided by PyMagnitude\(^5\), we settled to using the heavy GoogleNews with 300 dimensions (\texttt{GoogleNews-vectors-negative300.magnitude}). It produces good results and, at the same time, it is relatively easier to load and handle using pymagnitude toolkit. It takes between eight to ten minutes to build \texttt{ENT.vec} and \texttt{REL.vec} for a dataset with around 80K triplets. See fig 33 for the complete process of building those two models.

5.3.2.1 Example for building new KGE models (ENT/REL)

Starting by passing the knowledge graph dataset (csv file with 3 columns $h$, $r$, and $t$), we can build KGE for a new KG dataset either from the CLI or UI (streamlit interface).

\(^5\)https://github.com/plasticityai/magnitude#pre-converted-magnitude-formats-of-popular-embeddings-models
5.3 KGE QA System

Here is an example from command-line

```bash
$ python -m kgeqa.build_new_model -csv data/sample1_KG.csv
Started a model builder for data from: data/sample1_KG.csv
Building a new embedding model for 15 tokens ..
Done. See output: data/ENT.vec
Building a new embedding model for 7 tokens ..
Done. See output: data/REL.vec
Converting models to .magnitude format ..
Loading vectors... (this may take some time)
Found 15 key(s)
Each vector has 300 dimension(s)
Creating magnitude format...
Writing vectors... (this may take some time)
Successfully converted 'data/ENT.vec' to 'data/ENT.vec.magnitude'!
Successfully converted 'data/REL.vec' to 'data/REL.vec.magnitude'
Done.
```

Or from the user interface (UI). See fig 34.

---

**Building a new knowledge domain**

Enter the complete .csv file path of your knowledge graph (FORMAT: three columns h r t)

```
/Users/Aziz/Dropbox/thesis/code/kge_qa/lab/all_domains_cleaned.csv
```

Creating a new knowledge domain from:
```
/Users/Aziz/Dropbox/thesis/code/kge_qa/lab/all_domains_cleaned.csv
```

Building KGE embedding models ...

Figure 34: Example for building new KGE models from the UI

---

**Figure 33: Workflow for building ENT.vec and REL.vec models with an example**

**Figure 34: Example for building new KGE models from the UI**
Stdout logs for creating new KGE models from the UI:

```
Started a model builder for data from:
/Users/Aziz/Dropbox/thesis/code/kge_qa/lab/all_domains_cleaned.csv
Building a new embedding model for 7636 tokens ..
Done. See output: data/ENT.vec
Building a new embedding model for 32 tokens ..
Done. See output: data/REL.vec
Converting models to .magnitude format ..
Loading vectors... (this may take some time)
... Cleaning up temporary files...
Successfully converted 'data/ENT.vec' to 'data/ENT.vec.magnitude'!
Each vector has 300 dimension(s)
... Cleaning up temporary files...
Successfully converted 'data/REL.vec' to 'data/REL.vec.magnitude'!
Done
```

### Building a new knowledge domain

Enter the complete .csv file path of your knowledge graph (FORMAT: three columns h r t)

```
/Users/Aziz/Dropbox/thesis/code/kge_qa/lab/all_domains_cleaned.csv
```

Creating a new knowledge domain from:

```
/Users/Aziz/Dropbox/thesis/code/kge_qa/lab/all_domains_cleaned.csv
```

Building KGE embedding models ...

Done. Created new models:

```
data/ENT.vec, data/ENT.vec.magnitude, data/REL.vec, data/REL.vec.magnitude
```

**Reboot KGE QA system to load the new domain**

Figure 35: Created new embedding models for entities and relations

Description of the generated models:

- **data/ENT.vec** Entity embeddings in .txt format
- **data/ENT.vec.magnitude** Entity embeddings in PyMagnitude format
- **data/REL.vec** Relation embeddings in .txt format
- **data/REL.vec.magnitude** Relation embeddings in PyMagnitude format

### 5.3.3 System description and design

When it comes to natural language, different people may ask differently about the same thing. In other words, there are more than one way to ask the same question. For example,

- a specific fact
5.3 KGE QA System

• we want to know it, so we ask about it
• but, the question might comes in various forms,
• these variations are added complexity to any question answering system
• one of the key strengths of our QA system is its ability to deal with such variations and eliminate their factor of complexity\textsuperscript{58}.

As mentioned, a key characteristic of our Q/A framework is its capability to handle all the possible variations of the same question. In other words, any specific fact-based question (related to the domain knowledge) might be asked in various (natural language) forms. Yet, the system should still be able to address “subsume” those variations and get to the correct answer. For example, in Fig. 36, the first three questions are asking about See the questions 1, 2, and 3 in Fig. 36.

Figure 36: Example of CLI interface for the QA system. Same question can be asked in different ways

Question rules and assumptions

There can be two kinds of fact based questions, open-domain and closed-domain questions. As discussed earlier, we focus on the latter type where the expected questions fall under a specific domain. The subsequent section of this dissertation describes our domain in details. Also, in literature, there exists various datasets related to such kinds of questions\textsuperscript{59}.

To ask a closed-domain question, a question can be expressed in any form in English language. However, we define three rules or conditions that should be present in any input question:

\textsuperscript{58}This key strength idea was inspired by a separate, but similar, project that I worked on with a team of two other people (Yuecheng Zhu and Witold Szejgis) during our machine learning fellowship at fellowship.ai

\textsuperscript{59}for example: see wikidata, factoid-questions dataset, and BuboQA
1. It needs to be a factoid-based i.e. its answer is a concise fact. For example, “what is the capital city of Iceland?”

2. It has to be about knowledge contained within the domain knowledge of our training data.60

3. It needs to contain: 1) at least one entity (or its equivalent meaning) and 2) one relation (or its equivalent meaning) from our dictionary of the KG triplets.

System design

See the fig. 37.

5.3.3.1 Extracting Entities and Relations from question

Our goal is to identify any true entity or relation mentioned in the input question; True means exists

---

60More on this in section 5.3.5.1
in our embedding model. If we think of a question as a sequence of tokens: \texttt{TOKEN1 TOKEN2 TOKEN3} ..., we want to label each token’s type as either \texttt{<ENTITY>}, \texttt{<RELATION>}, or \texttt{<OTHER>}. We do that based on the token’s closeness to either \texttt{ENT.vec}, or \texttt{REL.vec} models.

For example, in the question “Who wrote the movie Troy?” we would start with the raw tokens ['who', 'wrote', 'the', 'movie', 'troy']. In order to minimize the complexity, we filter out the tokens by removing stop words and concatenating True subwords. True subwords means when a bigram of tokens is a True key in either \texttt{ENT.vec} or \texttt{REL.vec} models; such as in the case of ['directed', 'by'] becomes ['directed_by'] since directed_by exists in \texttt{REL.vec}. So in that example we end up with the filtered tokens ['wrote', 'movie', 'troy'].

Then, for each token, we find its most similar entity from \texttt{ENT.vec} and its most similar relation from \texttt{REL.vec} e.g. \texttt{model.most_similar(token)}. The similarity is calculated by taking the cosine distance between the vector of \texttt{token} and each vector in the embedding models. For instance closest entity to \texttt{wrote} is ('capote', 0.68), while the closest relation is ('written_by', 0.92); where 0.68 is the cosine similarity between the embedding vector of \texttt{wrote} and the vector of \texttt{capote}. In which we say the token \texttt{wrote} is most likely to be a \texttt{RELATION} since it is closer to \texttt{written_by}.

After all tokens are labeled, we end up with the following:

\texttt{LABELED TOKENS:}

\texttt{[('wrote', '<RELATION>'), ('movie', '<ENTITY>'), ('troy', '<ENTITY>')]}

During the labeling process, each token will be associated with its closest neighboring token from our (\texttt{ENT.vec}/\texttt{REL.vec}) models. For instance, \texttt{wrote} will be associated with \texttt{written_by}. To match with the true triplets in our KGE model, each of the input tokens will be swapped (replaced) to its closest true token. So, in that example, we will have the following:

\texttt{SWAPPED TOKENS:}

\texttt{[('wrote', 'written_by'), ('movie', 'epic_movie'), ('troy', 'troy')]}

Then at this stage, we have the following labeled tokens:

\texttt{LABELED TOKENS:}

\texttt{[('written_by', '<RELATION>'), ('epic_movie', '<ENTITY>'), ('troy', '<ENTITY>')]}

In the next step, we use the labeled tokens to form all the possible ENT-REL pairs as follows:

\texttt{CANDIDATE PAIRS:}

\texttt{[('epic_movie', 'written_by'), ('troy', 'written_by')]}
We then choose the true pair based on its existence in our domain knowledge dataset. That is:

SELECTED PAIR:
(troy, written_by)

Finally, and to find the answer, we need to complete the incomplete triplet above. To do so, we use our knowledge graph dataset to retrieve the tail(s) of any triplet that match above (head, relation) pair.

5.3.3.2 Determining Tokens Type

Given an input TOKEN we decide its type based on its closest neighboring vector from ENT.vec and REL.vec as below. First find the closest neighbors from both models:

\[ \text{neighbors} = \text{closest\_entities(token)} + \text{closest\_relations(token)} \]  

(12)

Where closest_entities is just the model.most_similar(token). Then, we want to decide which one of the neighbors is actually the closest as follows:

\[ \text{closest\_neighbor} = \begin{cases} \max(\text{similarity(neighbors)}), & \text{if similarity} \geq \text{MAX\_CONFIDENCE} \\ \max(\frac{\text{SM(neighbors)}+\text{similarity(neighbors)}}{2}), & \text{Otherwise} \end{cases} \]  

(13)

Where MAX\_CONFIDENCE and MIN\_CONFIDENCE (in 14) are thresholds. And similarity is CosineSimilarity and SM is SequenceMatcher. Setting values for the thresholds can be tricky, so we need to pick those values carefully. Based on our trial and error experiments, we use the values 0.9 and 0.2 respectively.

Finally, we decide the token type as in 14 below:

\[ \text{token\_type} = \begin{cases} \text{type(closest\_neighbor)}, & \text{if similarity(closest\_neighbor)} \geq \text{MIN\_CONFIDENCE} \\ \text{OTHER}, & \text{otherwise} \end{cases} \]  

(14)

5.3.3.3 Workflow of the Answering Process

The following figure 38 illustrates the complete workflow of the current answering task in KGE QA.
Figure 38: KGE QA workflow for answering a question
Advanced answering mechanism

Provided the input question, we assume the task as a link prediction task. In other words, after parsing the input, we will have one or more incomplete triplets. The goal, then, is to complete any missing part in a triplet of the form \((\text{Entity}_1, \text{Relation}, \text{Entity}_2)\). For instance, in the example query in 39, we have the given \((\text{TomCruise}, \text{played_role}, \text{FighterPilot})\) and our goal is to complete the unknown triplets:

\[ \text{Question: which movie did tom cruise play a fighter pilot?} \]

- \((\text{FighterPilot}, \text{role_in}, ?)\)
- \((\text{TomCruise}, \text{acted_in}, ?)\)

And match its resulted movie name(s) with the movie name that Tom Cruise have had acted in.

In our knowledge graph embedding, where entities and relationships are represented as a Vector Space Model VSM, the correct answer to the query translates to completing a simple arithmetic operation on the vectors of entities and relations:

- \(v_{\text{FighterPilot}} + v_{\text{role_in}} \approx V_{\text{candidates}}\)
- \(v_{\text{TomCruise}} + v_{\text{acted_in}} \approx V_{\text{candidates}}\)
**QUERY:** which movie did tom cruise play a fighter pilot?

**Answer:** Top Gun

![Conceptual approach for finding answers.](image)

Figure 40: Conceptual approach for finding answers.
5.3.4 System performance and accuracy

The overall performance depends on several factors. Generally, the cleanness of the knowledge graph dataset affects performance the most.

> HERE GOES the IRB experiment results with the analysis

5.3.5 KGE QA dataset description

Generally, the system is designed to work with any dataset as long as the dataset is structured in the correct format. A correct format is a list of comma-separated triples of facts with the header h,r,t. For example, assuming our knowledge graph dataset is a csv file (let’s say KG.csv), then it should like something like this:

\[
\begin{array}{ccc}
  h,r,t \\
  entity,relation,entity \\
  entity,relation,entity \\
  entity,relation,entity \\
  \ldots
\end{array}
\]

With that in mind, the system can be used with any customized dataset for any specific domain. However, for our testing purposes, we used a subset of a well known knowledge graph dataset called FB15K. See section 4.2.2 and the next section for more details about this dataset.

5.3.5.1 Description of the domain knowledge “Ontology”

The objective of this section is to describe the domain knowledge of our system. In other words, the domain and range for questions and their answers. This will make it easier to understand the kind of knowledge that our QA system is expected to cover. And ultimately, what kind of questions can be asked.

The SimpleQuestion dataset contains questions about FreeBase knowledge base. Each question is labeled with the correct answer and its corresponding triplet from FreeBase dataset. Initially, we thought of extracting a subset of these questions and consider them our target. However, we noticed that can be more rigid and difficult to scale. Therefore, instead, we decided to extract a large subset of FB15K dataset to be the knowledge domain of our system.

Understanding a FB15K triplet:
Before we dive deeper, let’s first recall what \textbf{FB15K} triplets look like. Here is an actual raw triplet example:

\[ /m/016h4r /\text{music/artist/genre} /m/026g51 \]

To simplify it, let’s label the three components of this triplet:

<table>
<thead>
<tr>
<th>head</th>
<th>relation</th>
<th>tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>/m/016h4r</td>
<td>/music/artist/genre</td>
<td>/m/026g51</td>
</tr>
</tbody>
</table>

Now, let’s convert each component to its corresponding meaning, as follows:

<table>
<thead>
<tr>
<th>triple part</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/m/016h4r</td>
<td>kristoffer kris_kristofferson american country music singer ...</td>
</tr>
<tr>
<td>/music/artist/genre</td>
<td>Subject: music, subtopic: artist, label: genre</td>
</tr>
<tr>
<td>/m/026g51</td>
<td>outlaw_country subgenre country music popular late ...</td>
</tr>
</tbody>
</table>

The entity descriptions are based on entity-to-word dataset published in [107]. The dataset is available on GitHub\footnote{Look for \texttt{entityWords.txt} file.}.

And simply put in English language, the above triplet would translate to:

Kris Kristofferson is an American singer who plays outlaw country music genre.

\textbf{Filtering FB15K and Understanding the Domain Ontology of the filtered FB15K dataset}

First, we started filtering out \textbf{FB15K} dataset by selecting the set of triplets that fall under the specific “domains” subjects that we intended to cover. The chosen subjects are Computer, Movie, Location, and People.

As follows:

- Computer
- Film
- People
- Location

The below table describe the statistics of the selected relations and triplets:
Table 20: stats of the filtered FB15K datasets

<table>
<thead>
<tr>
<th>Domain</th>
<th># of relations</th>
<th># of Triplets</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td>4</td>
<td>69</td>
</tr>
<tr>
<td>film</td>
<td>12</td>
<td>37394</td>
</tr>
<tr>
<td>people</td>
<td>10</td>
<td>30047</td>
</tr>
<tr>
<td>location</td>
<td>8</td>
<td>12588</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>34</strong></td>
<td><strong>80098</strong></td>
</tr>
</tbody>
</table>

In the following figures, we describe the graph ontology of each domain. We also provided sample questions for each domain.

![Computer ontology graph](image)

Figure 41: Computer ontology graph

Computer-domain sample questions:

- What languages were influenced by Java?
- What the software license of Linux OS?

Film-domain sample questions:

- Who directed The Matrix Movie?
- What movies were written by James Cameron?

People-domain sample questions:

- What language do people speak in Saudi Arabia?
- Where was Gandhi born?

Location-domain sample questions:

- What currency is used in Germany?
Figure 42: Film ontology graph

Figure 43: People ontology graph

Figure 44: Location ontology graph
5.3 KGE QA System

- What is the capital city of Australia?

Finally, here is the complete ontology of all the four domains together.

![Figure 45: Complete ontology graph of our QA system](image)

Extracting the relevant triplets from FB15K

Then, in order to specify the space of the type of questions that the factoid Q/A system accepts, we describe the domain ontology of the chosen domains.

In order to highlight the space (domain/range) of the questions, we provide an ontological description for the filtered dataset (domain knowledge).

Table 21: stats of the filtered relationships in FB15K

<table>
<thead>
<tr>
<th>filtering term</th>
<th>total avail. relations</th>
<th>selected relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>/computer/</td>
<td>24</td>
<td>4</td>
</tr>
</tbody>
</table>
### Filtering Terms and Available Relations

<table>
<thead>
<tr>
<th>Domain</th>
<th>Total Relations</th>
<th>Selected Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>/film/</td>
<td>95</td>
<td>12</td>
</tr>
<tr>
<td>/people/</td>
<td>47</td>
<td>10</td>
</tr>
<tr>
<td>/location/</td>
<td>63</td>
<td>8</td>
</tr>
</tbody>
</table>

Below is the complete list of the selected relations from each domain:

**computer** = [
  # FB15 total relations: 24
  # KEYWORD: /computer/
  '/computer/operating_system/parent_os', # 10
  '/computer/programming_language/influenced', # 26
  '/computer/programming_language/influenced_by', # 25
  '/computer/software/license', # 8
]

**film** = [
  # FB15 total relations: 95
  # KEYWORD: /film/
  '/film/film/starring/film/acting_role/actor', # 9466
  '/film/film/starring/film/acting_role/character', # 64
  '/film/film/genre', # 7263
  '/film/production_company/films', # 1537
  '/film/writer/film', # 807
  '/film/film/directed_by', # 1285
  '/film/film/written_by', # 850
  '/film/director/film', # 859
  '/film/film/produced_by', # 1285
]

**people** = [
  # FB15 total relations: 47
  # KEYWORD: /people/
  '/people/person/ethnicity', # 2030
  '/people/ethnicity/people', # 2073
  '/people/person/profession', # 11636
  '/people/person/place_of_birth', # 2468
  '/people/deceased_person/place_of_death', # 697
  '/people/person/education/education/institution', # 2591
  '/people/person/nationality', # 4198
  '/location/location/people_born_here', # 2485
  '/people/person/languages', # 783
  '/people/person/religion', # 1086
]

**location** = [
  # FB15 total relations: 63
  # KEYWORD: /location/
  '/location/location/containedby', # 5186
  '/location/location/contains', # 5204
  '/location/country/capital', # 142
  '/location/country/time_zones', # 1151
  '/location/country/languages_spoken', # 334
  '/location/country/official_language', # 225
  '/location/country/form_of_government', # 298
  '/location/country/currency_used', # 48
]
5.3.5.2 Visualizing Entities and Relations

As an additional feature, KGE QA system provides visualizations capability to display a graph of the closest Entities and/or Relations (from our ENT.vec/REL.vec models) to any word entered into the system. Since the embedding models are high-dimensional data, we used an external tool (t-SNE) to reduce the dimensionality of the vectors\(^62\).

See an example in the figure below.

Figure 46: Visualizing the closest 20 relations in our REL.vec to the word “wrote”

5.4 System setup and project configurations

System requirements

- Python3.6+
- pymagnitude==0.1.120
- streamlit==0.45.0
- pandas==0.25.3

Setup

First, let’s check out the source code from its git repository, create a virtual environment, and install the dependencies:

```
$ git clone https://github.com/iamaziz/kge_qa
$ cd kge_qa
$ virtualenv .env -p python3.7
$ source .env/bin/activate
(.env) $ pip install -r requirements.txt
```

Getting Started

The KGE QA system can be used either in the command-line mode or in a web browser. Open a terminal window to get started.

To run in command line mode, type:

```
(.env) $ python -m kgeqa.main
enter your question >>
```

Or to in the browser mode (requires streamlit), type:

```
(.env) $ streamlit run app.py
# go to: http://localhost:8501/ (see the screenshot below)
```

![KGE QA System](image)

Figure 47: localhost:8501 to get started in the browser
6 Conclusion

In this dissertation, we have introduced a new approach for building a representation model for knowledge graphs. Our method embed the knowledge graph into a low-dimensional vector space. This meaningful representation is essential for using knowledge graphs efficiently in various AI applications such as dialog systems. The presented approach is inspired by a combination of existing and successful techniques in natural language processing, machine learning, and relational knowledge representation.
7 Bibliography


