Template-Based Question Answering over Linked Data using Recursive Neural Networks

by

Ram Ganesan Athreya

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2018 by the
Graduate Supervisory Committee:

Srividya Bansal, Chair
Ricardo Usbeck
Kevin Gary

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

The Semantic Web contains large amounts of related information in the form of knowledge graphs such as DBpedia. These knowledge graphs are typically enormous and are not easily accessible for users as they need specialized knowledge in query languages (such as SPARQL) as well as deep familiarity of the ontologies used by these knowledge graphs. So, to make these knowledge graphs more accessible (even for non-experts) several question answering (QA) systems have been developed over the last decade. Due to the complexity of the task, several approaches have been undertaken that include techniques from natural language processing (NLP), information retrieval (IR), machine learning (ML) and the Semantic Web (SW). At a higher level, most question answering systems approach the question answering task as a conversion from the natural language question to its corresponding SPARQL query. These systems then utilize the query to retrieve the desired entities or literals. One approach to solve this problem, that is used by most systems today, is to apply deep syntactic and semantic analysis on the input question to derive the SPARQL query. This has resulted in the evolution of natural language processing pipelines that have common characteristics such as answer type detection, segmentation, phrase matching, part-of-speech-tagging, named entity recognition, named entity disambiguation, syntactic or dependency parsing, semantic role labeling, etc.

This has lead to NLP pipeline architectures that integrate components that solve a specific aspect of the problem and pass on the results to subsequent components for further processing eg: DBpedia Spotlight for named entity recognition, RelMatch for relational mapping, etc. A major drawback in this approach is error propagation that is a common problem in NLP. This can occur due to mistakes early on in the pipeline that can adversely affect successive steps further down the pipeline. Another approach is to use query templates either manually generated or extracted from existing benchmark datasets such as Question Answering over Linked Data (QALD) to generate the SPARQL queries that is basically a set of predefined queries with various slots that need to be filled. This approach potentially shifts the question answering problem into a classification task where the system needs to match the input question to the appropriate template (class label).

This thesis proposes a neural network approach to automatically learn and classify natural language questions into its corresponding template using recursive neural networks. An obvious advantage of using neural networks is the elimination for the need of laborious feature engineering that can be cumbersome and error prone. The input question would be encoded into a vector representation. The model will be trained and evaluated on the LC-QuAD Dataset (Large-scale Complex Question Answering Dataset). The dataset was created explicitly for machine learning based QA approaches for learning complex SPARQL queries. The dataset consists of 5000 questions along with their corresponding SPARQL queries over the DBpedia dataset spanning 5042 entities and 615 predicates. These queries were annotated based on 38 unique templates that the model will attempt to classify. The resulting model will be evaluated against both the LC-QuAD dataset and the Question Answering Over Linked Data (QALD-7) dataset.

The recursive neural network achieves template classification accuracy of 0.828 on the LC-QuAD dataset and an accuracy of 0.618 on the QALD-7 dataset. When the top-2 most likely templates were considered the model achieves an accuracy of 0.945 on the LC-QuAD dataset and 0.786 on the QALD-7 dataset.

After slot filling, the overall system achieves a macro F-score 0.419 on the LC-QuAD dataset and a macro F-score of 0.417 on the QALD-7 dataset.

DEDICATION

To my late father whose literal dying wish was that I do my Masters in the US and whose eternal doubts in me made me overcome my limitations time and time again.

ACKNOWLEDGMENTS

This work would not have been possible without the help, encouragement and opportunities provided by so many people. In no particular order I would like to thank,

Dr Srividya Bansal for being my Thesis Chair and getting me introduced to the Question Answering in Linked Data Challenge which got the ball rolling and finally result in this thesis. I would also like to thank her for helping me in seeking opportunities within ASU without which I would have found it very hard to support myself and focus on my research.

Dr Ricardo Usbeck for being an outstanding mentor during the Google Summer of Code Program (2017) as well as assisting and shepherding me throughout this thesis and graciously agreeing to be a part of my Thesis Committee.

Dr Kevin Gary for agreeing to be a part of my thesis committee. Also, I would not forget the accommodations you made during the Spring semester which allowed me to present my paper in WWW 2018 (WebConf).

Google and especially the **Google Summer of Code (GSoC)** program (especially **Stephanie Taylor**) which got me embedded within the DBpedia community and provided me the opportunity to contribute to open source development and build vital connections within DBpedia that were invaluable for my research as well as my career.

The **DBpedia** organization and community for supporting me throughout this work, especially **Sandra Prator** who was a great help during the GSoC process and our single point of contact regarding all administrative and logistical tasks, **Tim Ermilov**, **Paul Spooren** and **Felix Conrads** for infrastructure support, **Dennis**

Diefenbach and **Andreas Both** for all their help when I was in WWW 2018 in Lyon, and **Julia Holze** for ensuring that I was able to secure grants so that I could attend the conference.

Jaydeep Chakraborty for the passionate disagreements and stimulating conversations which were invaluable during the research process.

Dr Ajay Bansal for introducing me to Prolog and the fascinating world of declarative and answer set programming which I fervently hope will find its place in mainstream artificial intelligence someday.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

Diefenbach *et al.* (2017a) classify the techniques used in question answering (QA) systems (over linked data) broadly into five tasks:

1. **Question Analysis:** In this step, the question of the user is analyzed based on purely syntactic features. QA systems use syntactic features to deduce, for example, the right segmentation of the question, determine that phrase corresponds to an instance (subject or object), property or class and the dependency between the different phrases.

2. **Phrase Mapping:** This step starts with a phrase (one or more words) s and tries to find, in the underlying knowledge base (KB), a set of resources that correspond to s with high probability. s could correspond to an instance, property or a class from the knowledge base.

3. **Disambiguation:** Two ambiguity problems can arise. The first is that from the question analysis step the segmentation and the dependencies between the segments are ambiguous. For example, in the question "Give me all European countries." the segmentation can group or not group the expression "European countries" leading to two possibilities. The second is that, the phrase mapping step returns multiple possible resources for one phrase. In the example above "European" could map to different meanings of the word "Europe".

4. **Query Construction:** This phase deals with how the QA system constructs the SPARQL query to find the answer to the question. A problem arises during the query construction, that is commonly referred to as the "semantic gap". Assume for example that a user asks the question: "which countries are in the European Union?". One would probably assume that in the KB there are triples like:

   dbr:Italy dbo:member dbr:European_Union .

   dbr:Spain dbo:member dbr:European_Union .

   But this is not the case, in DBpedia the requested information is encoded as:

   dbr:Italy dct:subject dbc:Member_states_of_the_European_Union.

   dbr:Spain dct:subject dbc:Member_states_of_the_European_Union.

   where *dbr* is the <**http://dbpedia.org/resource/**> namespace, *dbo* is the <**http://dbpedia.org/ontology/**> namespace, *dct* is the <**http://dublincore.org/2012/06/14/dcterms**> namespace and *dbc* is the <**http://dbpedia.org/page/Category**> namespace. So instead of a property $dbo : member$ DBpedia uses the class $dbc : Member_{s}tates_{o}f_{t}he_{E}uropean_{U}nion$ to encode the information. The "semantic gap" refers to the problem that the KB encodes an information differently from what one could deduce from the question. This shows that in general it is difficult to deduce the form of the SPARQL query knowing only the question.

5. **Querying:** The final step is to query the underlying knowledge base to retrieve the answers for the given question. The answer can be from a single KB or depending on the system and the task even from multiple KBs.

Most question answering systems follow the above mentioned steps in the specified order. This places a difficulty on the query building process where multiple candidate templates can be generated for a sentence. Due to error propagation, that is mistakes in one step of the pipeline, can lead to crucial ramifications downstream and adversely affect the overall performance of the system.

This becomes especially difficult for complex queries that span multiple triples and many facts need to be discovered before the question can be answered. For complex questions, where the resulting SPARQL query contains more than one basic graph pattern, sophisticated approaches are required to capture the structure of the underlying query. Current research follows two paths, namely (1) template-based approaches, that map input questions to either manually or automatically created SPARQL query templates or (2) template-free approaches that try to build SPARQL queries based on the given syntactic structure of the input question. However, template-free approaches require additional effort of ensuring to cover every possible basic graph pattern making it a more computationally intensive process Höffner *et al.* (2017).

Template classification acts basically as an alternative to the query building approach or the sub-graph generation (from entities) approach that are more computationally intensive and error prone. Furthermore, as the analysis of Singh *et al.* (2018) on QALD subtasks shows, query building has one of the poorest F-Measures at 0.48. So, by performing template classification in the beginning, the workflow gets inverted and provides the benefit of restricting the number of resources, entities and ontology classes that need to be considered for a candidate SPARQL query instead of seemingly endless combinations as is usually done in a non-template approach.

For completeness, existing methods are used to fill the slots after template classification to provide a performance comparison against existing methods.

1.2 Problem Statement

The problem statement for the thesis can be summarized as follows:

1. Can state-of-the-art neural network techniques such as Long Short Term Memory (LSTM), recursive neural networks and word embeddings be leveraged for the template classification task?

2. Can a template classification model serve as a replacement for the query building process that has been shown to be both error-prone and computationally intensive Singh *et al.* (2018), Usbeck *et al.* (2015a), Saleem *et al.* (2017)?

3. Can the template classification model be developed without any domain specific information/features that can make it easily transferable across domains?

4. Can such a system exhibit reasonable performance on existing question answering challenges or datasets such as LC-QuAD and question answering over linked data (QALD-7) when compared to existing systems ?

Chapter 2

THE SEMANTIC WEB

2.1 Linked Data

Traditionally, data published on the Web has been made available as raw dumps in formats such as CSV or XML, or marked up as HTML tables, sacrificing much of its structure and semantics. In the conventional hypertext Web, the nature of the relationship between two linked documents is implicit, as the data format, i.e. HTML, is not sufficiently expressive to enable individual entities described in a particular document to be connected by typed links to related entities. The term Linked Data refers to a set of best practices for publishing and connecting structured data on the Web. These best practices have been adopted by an increasing number of data providers, leading to the creation of a global data space containing billions of assertions sometimes called the Web of Data. The Web of Data also opens up new possibilities for domain-specific applications. Unlike Web 2.0 mash-ups which work against a fixed set of data sources, Linked Data applications operate on top of an unbound, global data space. This enables them to deliver more complete answers as new data sources appear on the Web Bizer *et al.* (2011).

With this rapid growth of the Semantic Web (SW), the process of searching and querying content that is both massive in scale and heterogeneous have become increasingly challenging. User-friendly interfaces, that can support end users in querying and exploring this novel and diverse, structured information space, are needed to make the vision of the SW and Linked Data a reality Lopez *et al.* (2011).

2.2 Knowledge Graphs

A knowledge graph can be defined as a graph containing a set of assertions usually expressed as triples. The nodes of the graph are usually entities and the edges express relations between entities. Usually, knowledge graphs have a schema that is defined through a rigid ontology which encodes attributes of entities and relations. This allows capabilities such as reasoning on top of the knowledge base.

Wikipedia is the 6th most popular website, and the most widely used encyclopedia. There are official Wikipedia editions in over 280 different languages which range in size from a couple of hundred articles up to several million articles (English edition). Besides free text, Wikipedia articles consist of different types of structured data such as infoboxes, tables, lists, and categorization data. The DBpedia project builds a large-scale, multilingual knowledge base by extracting structured data from Wikipedia editions in 111 languages. This knowledge base can be used to answer expressive queries by leveraging the semantic information derived from Wikipedia. Being multilingual and covering a wide range of topics, the DBpedia knowledge base is also useful within further application domains such as data integration, named entity recognition, topic detection, document ranking, and question answering Lehmann *et al.* (2015).

2.3 SPARQL Protocol and RDF Query Language (SPARQL)

The data stored in a knowledge graph is in the form of triples that consist of three parts namely <Subject ><Predicate ><Object >. These triples are usually stored in a format called Resource Description Framework (RDF).

RDF is a W3C specification for storing and interchanging data on the Web. RDF extends the linking structure of the Web to use Uniform Resource Identifiers (URIs) to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

The two ends of a link are called resources that are defined using a Uniform Resource Identifier which can uniquely identify a resource or entity in the graph. The predicate is also a URI used to depict a relationship between two nodes in a knowledge graph.

SPARQL is an RDF query language geared towards manipulating and retrieving data stored in semantic databases or triplestores. SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, filtering and optional patterns for querying data.

Chapter 3

QUESTION ANSWERING OVER LINKED DATA

3.1 Introduction

Traditionally, question answering (QA) approaches have largely been focused on retrieving answers from raw text, with the emphasis on using ontologies to mark-up Web resources and improve retrieval by using query expansion McGuinness (2004).

Since the steady growth of the Semantic Web and the emergence of large-scale knowledge graphs, the necessity of natural language interfaces to ontology-based repositories has become more acute, re-igniting interest in question answering systems. This trend has also been supported by usability studies conducted by Kaufmann and Bernstein (2007), which show that casual users, typically overwhelmed by the formal logic of the Semantic Web, prefer to use a natural language interface to query an ontology.

Hence, in the past decade there has been much interest in ontology-based question answering systems, where the power of ontologies as a model of knowledge is directly exploited for the query analysis and translation. Typically, in such a question answering system over a knowledge base incoming queries are generally addressed by translating a natural question to a SPARQL query that can be used to retrieve the desired information Diefenbach *et al.* (2017a).

In the last number of years, different benchmarks for question answering systems over knowledge bases have been developed. The most popular among them in the Semantic Web community is the Question Answering Over Linked Data (QALD) dataset. In fact, QALD is not one benchmark but a series of evaluation campaigns for QA systems over knowledge bases. There have been 9 iterations of the challenge till date that have been conducted on an annual basis.

Another interesting dataset is the Large-Scale Complex Question Answering (LC-QuAD) dataset. Unlike QALD, LC-QuAD was developed from the ground up to facilitate machine learning based QA approaches. QALD is insufficient in terms of size, variety, or complexity which are essential when applying and evaluating neural network approaches. The dataset contains 5000 questions with their corresponding SPARQL queries and associated query templates which are 38 in number, Trivedi *et al.* (2017). For this reason LC-QuAD was chosen for training recursive neural network model for template classification task. Query templates basically present a blueprint of the final SPARQL query that needs to be derived from the question to find the corresponding answer(s).

3.2 Natural Language Processing Pipelines

The key QA tasks of most question answering systems comprise of Named Entity Recognition and Disambiguation, Relation Extraction and Query Building. No single system will be perfect for all tasks and across all domains. This has led to the development of QA components that specialize in specific tasks for specific domains which can then be bootstrapped into modular question answering pipelines.

Diefenbach *et al.* (2017b) developed QANARY, a message-driven and light-weight architecture that leverages linked data technology and particularly vocabularies to create a component-based QA system. Their approach solves a critical problem in the QA community, that is integrating existing components which is resource intensive process. They solve this problem through an RDF based 'qa' vocabulary which is a flexible and extensible data model for QA systems. The vocabulary enables composition and integration of QA components in such a way that it is independent from programming languages, agnostic to domains and datasets, as well as enabled for components on any granularity level within the QA process.

Singh *et al.* (2018) studied the efficiency of these components by training classifiers which take features of a question as input and have the goal of optimizing the selection of QA components based on those features. They then employ a greedy algorithm to identify the best pipeline that includes the best possible components which can effectively answer the given question. The system was evaluated using the QALD and LC-QuAD benchmarks. They discovered that among the available solutions for the three tasks in question answering Named Entity Recognition ranks the highest (based on Macro Precision, Recall and F-Score) followed by Query Building and finally Relation Linking.

3.3 Template Question Answering

Most question answering systems translate questions into triples which are matched against an existing knowledge base to retrieve an answer to the question by using either a similarity or ranking metric. However, in many cases, such triples do not accurately represent the semantic structure of the natural language question which can result in flawed SPARQL queries or wrong answers.

To circumvent this problem, Unger *et al.* (2012) proposed an approach that relies on a parse tree of the question to produce a SPARQL template that directly mirrors the internal structure of the question. This template contains empty slots which are then instantiated using statistical entity identification and predicate detection. Their approach generates multiple candidate templates for a given query based on linguistic analysis and then uses similarity measures to determine the best possible template that adequately represents the question.

Since the system needs to generate templates from scratch it relies on a lexicon which is a composite of domain dependent and domain independent expressions. In this approach it is not known beforehand which URIs these expressions should be mapped to. So instead, they contain slots which are built on-the-fly while parsing, based on part-of-speech information provided by the Stanford POS tagger Toutanova *et al.* (2003a), and uses a set of simple heuristics that specify which POS tag corresponds to which syntactic and semantic properties.

By contrast, in this work the recursive neural network would automatically learn the required representations through labeled examples provided in the LC-QuAD dataset and the methodology is essentially domain independent and can thus be transposed to work with any domain requiring very little additional modifications to the neural network architecture.

3.4 Slot Filling

The goal of Slot Filling is to extract predefined types of attributes or slots for a given query. The slots in the linked data context can be a resource, predicate or ontology class. One critical component of slot filling is relation extraction. Typically, question answering systems over knowledge graphs rely on entity and relation linking components in order to connect the natural language input to the underlying knowledge graph. Dubey *et al.* (2018) propose a framework called EARL, which performs entity linking and relation linking as a joint task. They use two strategies for solving this problem. The first is to use an instance of the Generalized Travelling Salesman Problem (GTSP) and the second approach is to use machine learning to exploit the connection density between nodes in the knowledge graph. The system was evaluated against the LC-QuAD dataset. Both strategies significantly outperform the current state-of-the-art approaches for entity and relation linking with the adaptive learning model showing slightly higher performance compared to the GTSP model.

3.5 Conclusion

This section summarizes the related work and available solutions and approaches in Question Answering Linked Data. The next section would cover an overview of the LC-QuAD dataset and how it was preprocessed to facilitate the template classification task.

Chapter 4

LC-QUAD DATASET

4.1 Introduction

An essential requirement to develop and evaluate question answering systems is the availability of a large dataset comprising of varied questions and their corresponding logical forms. LC-QuAD consists of 5,000 questions along with the intended SPARQL queries required to answer questions over DBpedia. The dataset includes complex questions, i.e. questions in which the intended SPARQL query does not consist of a single triple pattern.

Trivedi *et al.* (2017) generated the dataset by using a list of seed entities, and filtering by a predicate whitelist, generate subgraphs of DBpedia to instantiate SPARQL templates, thereby generating valid SPARQL queries. These SPARQL queries are then used to instantiate Normalized Natural Question Templates (NNQTs) which act as canonical structures and are often grammatically incorrect. These questions are manually corrected and paraphrased by reviewers. Fig 4.1 provides a pictorial overview of the LC-QuAD dataset generation process.

There are two key advantages for using LC-QuAD over similar existing datasets such as SimpleQuestions Bordes *et al.* (2015), Free917 Cai and Yates (2013), and QALD-6 Unger *et al.* (2016). They are:

1. Higher focus on complex questions unlike SimpleQuestions which focuses entirely on single triple patterns.

2. Larger volume and variety of questions. The Free917 dataset contains only 917 questions and QALD-6 has 450 training questions and 100 test questions.

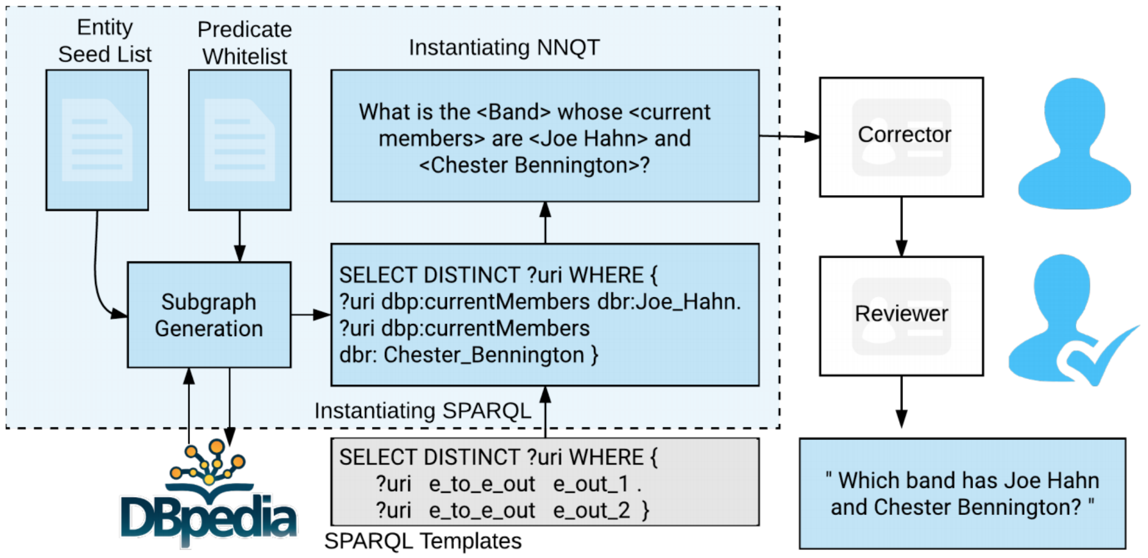

Entity
Seed List
Predicate
Whitelist
Instantiating NNQT
What is the <Band> whose <current members> are <Joe Hahn> and <Chester Bennington>?
Corrector
Subgraph
Generation
SELECT DISTINCT ?uri WHERE {
?uri dbp:currentMembers dbr:Joe_Hahn.
?uri dbp:currentMembers
dbr: Chester_Bennington }
Reviewer
Instantiating SPARQL
SELECT DISTINCT ?uri WHERE {
?uri e_to_e_out e_out_1 .
?uri e_to_e_out e_out_2 }
DBpedia
SPARQL Templates
" Which band has Joe Hahn and Chester Bennington? "

Figure 4.1: LC-QuAD Dataset Generation Workflow

4.2 Dataset Analysis

The LC-QuAD dataset contains 5,000 questions divided into 38 unique SPARQL templates comprising 5042 entities and 615 predicates. The SPARQL queries have been generated based on the 2016 DBpedia release. The dataset broadly contains three types of questions:

1. **Entity Queries:** Questions whose answer is an entity or list of entities with the WHERE clause containing one or more triples.

2. **Boolean Queries:** Questions whose answer is a boolean True or False with the WHERE clause containing exactly one triple.

3. **Count Queries:** Questions whose answer is a cardinal number with the WHERE clause containing one or more triples.

Among the 5000 verbalized SPARQL queries, only 18% are simple questions, and the remaining queries either involve more than one triple, or COUNT/ASK keyword, or both. Moreover, 18.06% queries contain a COUNT based aggregate, and 9.57% are boolean queries. The advantage of using LC-QuAD is that it was tailored specifically for neural network approaches to question answering and has a relatively large variety of questions in the complex, count and boolean categories when compared to existing datasets which is valuable when training models and evaluating approaches. As of now, the dataset does not have queries with OPTIONAL, or UNION keywords. Also, it does not have conditional aggregates in the query head Trivedi *et al.* (2017).

Table 4.1 displays the frequency distribution of each template in the LC-QuAD dataset along with its corresponding SPARQL template and an example query. Interestingly, the first 14 templates make up over 80% of the dataset and there are 7 templates with under 10 examples. In fact, templates 601, 9 and 906 have only 1 example in the entire dataset.

Table 4.1: Frequency Distribution of Templates in LC-QuAD Dataset

ID	**Count**	**Question Type**	**SPARQL Template**	**Example Query**
2	748	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?uri . }	Name the mascot of Austin College ?
305	564	Entity	SELECT DISTINCT ?uri WHERE { ?x <p ><r >. ?x <p2 >?uri . ?x rdf:type <class >. }	What layout can be found in cars similar to the Subaru Outback?

16	523	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?uri. <r2 ><p2 >?uri . }	Which series has an episode called The lost special and also a character named Sherlock Holmes ?
308	334	Entity	SELECT DISTINCT ?uri WHERE {?uri <p ><r >. ?uri <p2 ><r2 >. ?uri rdf:type <class >}	Name the mountain whose range is Sierra Nevada (U.S.) and parent mountain peak is Nevado de Toluca?
301	309	Entity	SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. ?uri rdf:type <class >}	What is the river whose mouth is in deadsea?
3	262	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?x . ?x <p2 >?uri . }	What awards did the film director of The Haunted House win ?
5	213	Entity	SELECT DISTINCT ?uri WHERE { ?x <p ><r >. ?x <p2 >?uri . }	Starwood operates in which places?

15	198	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?uri. <r2 ><p >?uri . }	In which part of the world can i find Xynisteri and Mavro?
152	188	Boolean	ASK WHERE { <r ><p ><r2 >. }	Was Ganymede discovered by Galileo Galilei?
151	180	Boolean	ASK WHERE { <r ><p ><r2 >. }	Does the Toyota Verossa have the front engine design platform?
306	175	Entity	SELECT DISTINCT ?uri WHERE { ?x <p ><r >. ?uri <p2 >?x . ?uri rdf:type <class >}	Which newspapers are owned by companies which are under Rolv Erik Ryssdal?
105	101	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?x <p ><r >. ?x <p2 >?uri . }	How many awards have been given to screenwriters?
1	159	Entity	SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. }	What are the beverages whose origin is England?

303	115	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?x . ?x <p2 >?uri . ?x rdf:type <class >}	What is the region of the ethnic group which speaks the language of Arkansas?
6	94	Entity	SELECT DISTINCT ?uri WHERE { ?x <p ><r >. ?uri <p2 >?x . }	What are some characters of the series produced by Ricky Gre-vais?
405	90	COUNT	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?x <p ><r >. ?x <p2 >?uri . ?uri rdf:type <class >}	How many companies have launched their rockets from the Vandenerg Air base?
401	77	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?uri <p ><r >. ?uri rdf:type <class >}	How many places were ruled by Eliza-beth II?

111	76	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?x <p ><r >. ?x <p >?uri }	Count the number of sports played by schools which play hockey ?
311	76	Entity	SELECT ?uri WHERE { ?x <p ><r >. ?x <p >?uri . ?x rdf:type <class >}	Name all the doctoral student of the scientist who also supervised Mary Ainsworth ?
406	70	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?x <p ><r >. ?uri <p2 >?x . ?uri rdf:type <class >}	How many TV show has distributor located in Burbank California ?
307	69	Entity	SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. ?uri <p ><r2 >. ?uri rdf:type <class >}	What is the river that falls into North Sea and Thames Estuary?
101	67	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?uri <p ><r >. }	How many movies did Stanley Kubrick direct?

7	62	Entity	SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. ?uri <p ><r2 >}	Whose former teams are Indianapolis Colts and Carolina Panthers?
8	33	Count	SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. ?uri <p2 ><r2 >. }	Which colonel consort is Dolley Madison?
102	26	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { <r ><p >?uri }	How many states does the Pioneer corporation operate in?
106	22	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?x <p ><r >. ?uri <p2 >?x . }	Count all those whose youth club was managed by Luis Enrique.
11	20	Entity	SELECT ?uri WHERE { ?x <p ><r >. ?x <p >?uri . }	List the outflows of the lake which has Benu river as one of it ?

403	17	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { <r ><p >?x . ?x <p2 >?uri . ?x rdf:type <class >}	How many countries surround the sea into which the Upper Neratva flow?
103	17	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { <r ><p >?x . ?x <p2 >?uri . }	How many other important things have been written by the creator of Stuart Alan Jones?
108	14	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?uri <p ><r >. ?uri <p2 ><r2 >. }	How many bacteria have taxonomy as Bacillales and domain as Bacteria?
315	10	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?uri. <r2 ><p >?uri . ?uri rdf:type <class >}	Which city is the resting place of the Martin Ragaway and Chuck Connors ?
402	9	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { <r ><p >?uri . ?uri rdf:type <class >}	How many teams was Garry Unger in, previously?

316	5	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?uri . <r2 ><p2 >?uri . ?x rdf:type <class >}	List the people casted in Betsy's Wedding and 16 candles?
107	5	Count	SELECT DISTINCT COUNT(?uri) WHERE { ?uri <p ><r >. ?uri <p ><r2 >. }	Count the number of shows whose creators are Jerry Seinfeld and Larry David?
605	2	Entity	SELECT DISTINCT ?uri WHERE { ?x <p ><r >. ?x <p2 >?uri . ?x rdf:type <class >}	What are the kind of games one can play on windows?
601	1	Entity	SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. ?uri rdf:type <class >}	Which technological products were manufactured by Foxconn?
9	1	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?x . ?x <p >?uri . }	Who is owner of the soccer club which owns the Cobham Training Centre?

906	1	Entity	SELECT DISTINCT ?uri WHERE { ?x <p ><r >. ?uri <p2 >?x . ?uri rdf:type <class >}	Name some TV shows whose theme is made by a band associated with Buckethead?

4.3 Data Preprocessing

As shown in Table 4.1 from the previous section there is great imbalance between the distribution of templates in the dataset. Also, some templates are exact replicas of others with an additional triple. For example, templates below 100 and templates in the 3xx series and templates in the 1xx and 4xx series have only one triple differentiating them:

?var rdf:type <class>

With this in mind, during preprocessing all templates which had less than 50 examples in the initial dataset were removed. The rationale here was that each template should have at least a 1% representation in the final dataset. Also, templates below 100 were merged with their corresponding 3xx templates and 1xx templates were merged with 4xx templates by adding additional OPTIONAL queries to the SPARQL template. Also, templates 151 and 152 were merged into each other since they have identical SPARQL templates.

For example template 1 and template 301 were combined into a single template as follows:

Template 1: **SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. }**

Template 301: **SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. ?uri rdf:type <class >}**

Combined Template: **SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. OPTIONAL { ?uri rdf:type <class >} }**

The removal of sparse templates resulted in only 80 questions being removed and the final dataset had 4,920 questions spread across 15 templates. The frequency distribution and updated templates of the preprocessed dataset are shown in Table 4.2. It must be noted that this refined dataset was used to train the template classification model. In spite of the manual review process there were several grammatical mistakes and misspellings of proper nouns in the dataset which were corrected as needed and the results of the same is shared with the LC-QuAD team so that they can improve the quality of the dataset for the community.

Table 4.2: Frequency of templates after preprocessing. Templates with <50 examples removed and similar templates merged

ID	**Templates Merged**	**Count**	**Question Type**	**New SPARQL Template**
5	5, 305	777	Entity	SELECT DISTINCT ?uri WHERE { ?x <p ><r >. ?x <p2 >?uri . OPTIONAL { ?x rdf:type <class >} }
2	2	748	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?uri . }
16	16	523	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?uri . <r2 ><p2 >?uri . }

1	1, 301	468	Entity	SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. OPTIONAL { ?uri rdf:type <class >} }
3	3, 303	377	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?x . ?x <p2 >?uri . OPTIONAL { ?x rdf:type <class >} }
151	151, 152	368	Boolean	ASK WHERE { <r ><p ><r2 >. }
8	308	334	Entity	SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. ?uri <p2 ><r2 >. ?uri rdf:type <class >}
6	6, 306	269	Entity	SELECT DISTINCT ?uri WHERE { ?x <p ><r >. ?uri <p2 >?x . OPTIONAL { ?uri rdf:type <class >} }
105	105, 405	261	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?x <p ><r >. ?x <p2 >?uri . OPTIONAL { ?uri rdf:type <class >} }
15	15	198	Entity	SELECT DISTINCT ?uri WHERE { <r ><p >?uri. <r2 ><p >?uri . }
101	101, 401	144	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?uri <p ><r >. OPTIONAL { ?uri rdf:type <class >} }
7	7, 307	131	Entity	SELECT DISTINCT ?uri WHERE { ?uri <p ><r >. ?uri <p ><r2 >. OPTIONAL { ?uri rdf:type <class >} }
111	111	76	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?x <p ><r >. ?x <p >?uri }

11	311	76	Entity	SELECT ?uri WHERE { ?x <p ><r >. ?x <p >?uri . ?x rdf:type <class >}
106	406	70	Count	SELECT (COUNT(DISTINCT ?uri) as ?count) WHERE { ?x <p ><r >. ?uri <p2 >?x . ?uri rdf:type <class >}

Chapter 5

QUESTION ANALYSIS AND TEMPLATE CLASSIFICATION

5.1 Introduction

Diefenbach *et al.* (2017a) classify the techniques used in question answering (QA) systems broadly into five tasks:

1. Question Analysis

2. Phrase Mapping

3. Disambiguation

4. Query Construction

5. Querying

It must be noted that not all question answering systems follow the above steps in the exact same order and in some systems the steps could be merged together into a single step. The proposed system follows the steps mentioned below:

1. Question Analysis

2. Template Classification (Query Construction)

3. Slot-Filling (Phrase Mapping and Disambiguation)

4. Querying

The first two steps are handled in this chapter while the last two are elaborated in the next chapter. This is because the output from step 1 is directly used in step 2 and the same is true for steps 3 and 4.

5.2 Question Analysis

In this step, the question of the user is analyzed based on purely syntactic features. QA systems use syntactic features to deduce, for example, the right segmentation of the question, determine which phrase corresponds to an instance (subject or object), property or class and the dependency between the different phrases Diefenbach *et al.* (2017a).

For now, we only deal with syntactic parsing of the incoming question in this phase and converting it into a form that can be used for training the Recursive Neural Network.

5.2.1 Part of Speech Tagging

Part of Speech (POS) Tagging is the process of annotating a word in a text as corresponding to a particular part of speech eg: noun, verb, adjective, etc. In Natural Language Processing (NLP) applications, POS tagging is usually the first step in a pipeline and the output of POS tagging is typically used by downstream processes such as parsing for instance.

For the model, the English version of the Stanford POS Tagger was used Toutanova *et al.* (2003b). The Stanford POS Tagger is a log-linear POS tagger which utlilizes both preceding and following tag contexts through the implementation of a dependency network representation. It also uses a broad set of lexical features, including joint conditioning on multiple consecutive words and fine-grained modeling of unknown word features. The tagger uses the Penn Treebank Tagset Marcus *et al.* (1993) for tagging the individual parts of speech and the Java implementation (v3.9.1) of the tagger was used.

For example, consider the question *"Philadelphia City Council is the governing* *body of which city ?"*. The corresponding POS tagged question is represented in Fig 5.1.

5.2.2 Dependency Parsing

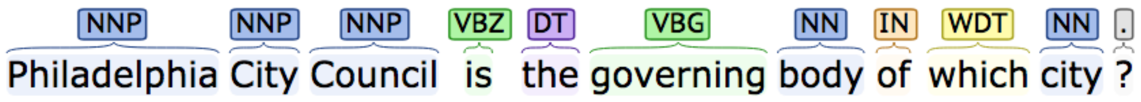
Parsing in NLP is the process of determining the syntactic structure of text using a formal grammar. A formal grammar is basically a formal description of the structure of a language. Given a sentence, a parser computes the combination of production rules that generate the sentence according to the underlying grammar.

POS tagged information alone is not enough to identify the relationships between the different chunks in a question. But this information can be leveraged by parsers to provide rich meaningful information between constituent words.

The Stanford Neural Network dependency parser was used by the system Chen and Manning (2014). The input to the parser was the sequence of POS tags generated from the previous step and the output is the corresponding parse tree. The parser uses low-dimensional, dense word embeddings which can effectively alleviate sparsity by sharing statistical strength between similar words, which results in a fast neural network classifier that can make parsing decisions within a transition-based dependency parser. The Java implementation (v3.9.1) on the Stanford parser was used by the system.

Fig 5.2 represents the Stanford Dependency Parser output for the question *"Philadelphia City Council is the governing body of which city ?"*.

There are two outputs of interest from Dependency Parsing. The first is the


NNP
NNP
NNP
VBZ
DT
VBG
NN
IN
WDT
NN
.
Philadelphia City Council is the governing body of which city ?
Figure 5.1: Stanford POS-Tagger Output

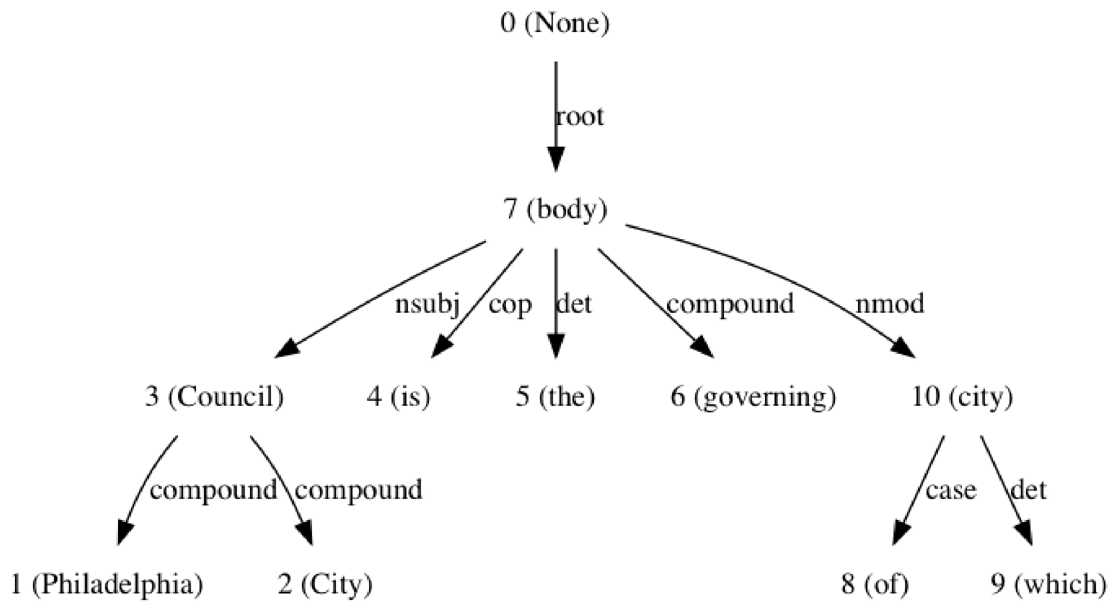


Figure 5.2: Stanford Dependency Parser Output

typed dependencies for each word in the input sentence. The typed dependencies representation provide a simple description of the grammatical relationships in a sentence. Its expressed as triples of a relation between pairs of words. For the rest of the thesis these relationships are denoted as RELS. The second output is the parse tree. Typically, a parse tree is the syntactic representation of a sentence based on a context-free grammar. It must be noted that a dependency parse tree does not make a distinction between terminal and non-terminal categories. Also, they are simpler on average than constituency-based parse trees because they contain fewer nodes.

The second output from the Stanford Parser is the dependency parse tree which is in parent pointer format. A parent pointer tree is an N-ary tree data structure in which each node has a pointer to its parent node but no pointers to child node. The structure can be visualized as a set of singly linked lists that share part of their structure, specifically their tails. From a given node, it is possible to traverse to the ancestors of that node but not to any other node. This allows for sequential processing of a parse tree which would allow efficient training of the template classification model.

5.3 Input Preparation

The output from the parse tree needs to be vectorized so that they can be supplied to the neural network model. There are two strategies to vectorize words. They are:

1. One-Hot Encoding

2. Word Embedding

Based on the available input data five distinct kind of input models were developed for training. These are summarized in Table 5.1

5.3.1 One-Hot Encoding

One-Hot encoding is a common strategy in machine learning for converting categorical input into a vector by setting all values as 0 except for 1 bit which has a value 1, hence the name One-Hot. For example the number of POS tags in the LC-QuAD dataset is 43. So each POS tag is represented as a 43 x 1 vector where a single index is 1 and the rest are 0 depending on the index of the POS tag being considered. This conversion from an abstract categorical value to a consistently sized vector enables easier processing and prediction by machine learning models.

5.3.2 Word Embedding

Generally, the goal of word embeddings is mapping the words in unlabeled text data to a continuously-valued low dimensional space, in order to capture the internal semantic and syntactic information. The concept of word embedding was first introduced with the Neural Networks Language Model (NNLM). They are usually unsupervised models and incorporate various architectures such as Restricted Boltzmann

Machine (RBM), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and Long-Short Term Memory (LSTM) that can be used to build word embeddings. Usually the goal of the NNLM is to maximize or minimize the function of log likelihood, sometimes with additional constraints Li and Yang (2018). A key reason for using word embedding is that, in the past few years it has been shown that pre-trained models produce vastly better performance compared to existing methods such as one-hot vectors.

Facebook FastText

For word embedding the system uses Facebook's FastText embedding model Bojanowski *et al.* (2016). FastText uses an approach based on the skipgram model (taking into account subword information), where each word is represented as a bag of character n-grams. A vector representation is associated to each character n-gram; words being represented as the sum of these representations. The main advantage of using FastText is its ability to handle out of vocabulary words better due to the use of the skipgram model. The dataset had over 6000 unique tokens which were compressed into word vectors of dimensions 300 x 1 using the FastText word embedding model.

Table 5.1: Dimensionality of different models created for the template classification task

Model	**Dimensionality**	**Description**
POS	43 x 1	Only POS Tags expressed as One-Hot Vector

POS + RELS	85 x 1	One-Hot POS vector concatenated with One-Hot RELS[1] Vector
FastText	300 x 1	FastText Word Embedding
FastText + POS + RELS	385 x 1	FastText Word Vector concatenated with One-Hot POS and RELS Vector
FastText + POS + RELS + CHARS	444 x 1	FastText Word Vector concatenated with One-Hot POS, RELS and CHARS[2] Vector

5.4 Recursive Neural Network

A recursive neural network is basically an extension of a recurrent neural network implemented on a graph or tree-based input instead of a sequential input. Subsequent sections deal with a basic introduction on recurrent neural networks and the Long Short Term Memory (LSTM) cell which would lay the groundwork for extension into the Tree-LSTM recursive neural network architecture which was used to train the model for the proposed system.

5.4.1 Recurrent Neural Networks and LSTM

Recurrent neural networks (RNNs) are able to process input sequences of arbitrary length via the recursive application of a transition function on a hidden state vector h_t. At each time step t, the hidden state h_t is a function of the input vector x_t that the

[1]RELS denote the relationships derived from the Dependency Parse of all questions. eg: nsubj, pobj, etc. There were 42 unique RELS tags in the dataset.

[2]The CHARS vector for a word is the average of One-Hot Vectors of the characters of each word in the question.

network receives at time t and its previous hidden state $h_{t\text{-}1}$. For example, the input vector x_t could be a vector representation of the t-th word in a body of text Elman (1990), Mikolov (2012). The hidden state $h_t \epsilon \mathbb{R}^d$ which represents a d-dimensional distributed representation of the tokens observed up to time t.

Commonly, the recurrent neural network transition function is an affine transformation followed by a pointwise nonlinearity such as the hyperbolic tangent function:

$$h_t = tanh(Wx_t + Uh_{t-1} + b) \tag{5.1}$$

where h_t is the new hidden state at time step t, h_{t-1} is the previous hidden state at time step $t-1$, x_t is the input at time step t. Finally, W, U and b are parameter vectors that the model is trying to learn.

Unfortunately, a problem with recurrent neural networks with transition functions of this form is that during training, components of the gradient vector can grow or decay exponentially over long sequences which is called the vanishing gradient problem Hochreiter (1998).

To address this problem, Hochreiter and Schmidhuber (1997) proposed the LSTM architecture which mitigates the issue of learning long-term dependencies by introducing a memory cell that is able to preserve state over long periods of time. While numerous LSTM variants have been described, here we describe the version used by Zaremba and Sutskever (2014).

The LSTM unit at each time step t can be defined as a collection of vectors in $\mathbb{R}^d$, an input gate i_t, a forget gate f_t, an output gate o_t, a memory cell c_t and a hidden state h_t. The values of the gating vectors i_t, f_t and o_t are in [0, 1] and d is the memory dimension of the LSTM.

The LSTM transition equations are as follows:

$$\begin{aligned}
i_t &= \sigma(W^{(\mathrm{i})}x_t + U^{(\mathrm{i})}h_{t-1} + b^{(\mathrm{i})}),\\
f_t &= \sigma(W^{(\mathrm{f})}x_t + U^{(\mathrm{f})}h_{t-1} + b^{(\mathrm{f})}),\\
o_t &= \sigma(W^{(\mathrm{o})}x_t + U^{(\mathrm{o})}h_{t-1} + b^{(\mathrm{o})}),\\
u_t &= \sigma(W^{(\mathrm{u})}x_t + U^{(\mathrm{u})}h_{t-1} + b^{(\mathrm{u})}),\\
c_t &= i_t \odot u_t + f_t \odot c_{t-1},\\
h_t &= o_t \odot tanh(c_t)
\end{aligned} \tag{5.2}$$

where x_t is the input at the current time step, σ denotes the logistic sigmoid function and $\odot$ denotes elementwise multiplication. The forget gate controls the extent to which the previous memory cell is forgotten, the input gate controls how much each unit is updated, and the output gate controls the exposure of the internal memory state. The hidden state vector in an LSTM unit is therefore a gated, partial view of the state of the unit's internal memory cell. Since the value of the gating variables vary for each vector element, the model can learn to represent information over multiple time scales.

5.4.2 Tree-LSTM

Recursive neural networks are non-linear adaptive models that are able to learn deep structured information. They were introduced as promising machine learning models for processing data from structured domains. They can be employed for both classification and regression problems and are capable of solving both supervised and unsupervised tasks. They provide the flexibility of being able to work with input of arbitrary length compared to other feature based approaches which are constrained to fixed length vectors Chinea (2009).

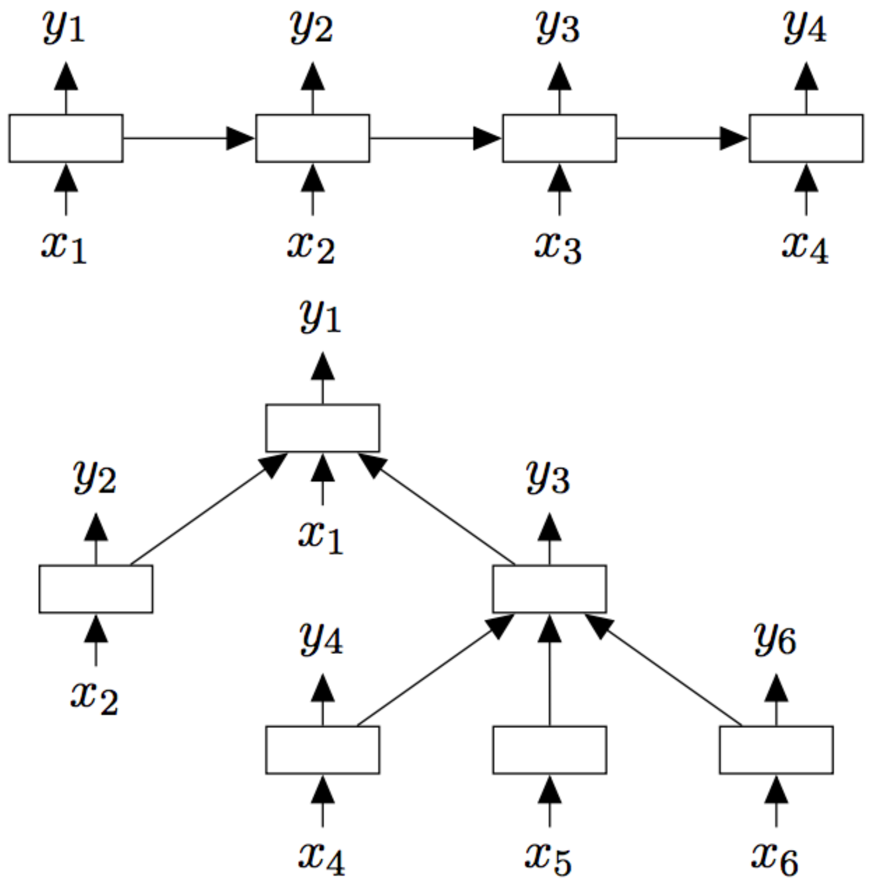


Figure 5.3: Top: A chain-structured LSTM network (Recurrent Neural Network) **Bottom:** A tree-structured LSTM network with arbitrary branching factor (Recursive Neural Network) Tai *et al.* (2015)

The Tree-LSTM was implemented based on the model proposed by Tai *et al.* (2015). The architecture presented below is based on their implementation. Tree-LSTM is a generalization of LSTMs to tree-structured network topologies. A key distinction between Tree-LSTM and standard LSTM is that, while the standard LSTM composes its hidden state from the input at the current time step and the hidden state of the LSTM unit in the previous time step, the tree-structured LSTM, or Tree-LSTM, composes its state from an input vector and the hidden states of arbitrarily many child units. The standard LSTM can then be considered a special case of the Tree-LSTM where each internal node has exactly one child. Fig 5.4.2 pictorially depicts this distinction between the architectures of the two models. For the proposed system, the Child-Sum Tree-LSTM architecture was used.

Similar to standard LSTM units, each Tree-LSTM unit (indexed by j) contains input and output gates i_j and o_j, a memory cell c_j, hidden state h_j and input vector x_j where x_j is a vector representation of a word in a sentence. The critical difference between the standard LSTM unit and Tree-LSTM units is that gating vectors and memory cell updates for a given node are dependent on the states of its child units. Additionally, instead of a single forget gate, the Tree-LSTM unit contains one forget gate f_{jk} for each child k. This allows the Tree-LSTM unit to selectively incorporate information from each child. For example, a Tree-LSTM model can learn to emphasize semantic heads in a semantic relatedness task, or it can learn to preserve the representation of sentiment-rich children for sentiment classification Tai *et al.* (2015).

Given a tree, let $C(j)$ denote the set of children of node j. The Tree-LSTM transition equations are the following:

$$\widetilde{h_j} = \sum_{k \epsilon C(j)} h_k \tag{5.3}$$

$$i_j = \sigma(W^{(\mathrm{i})} x_j + U^{(\mathrm{i})} \widetilde{h_j} + b^{(\mathrm{i})}) \tag{5.4}$$

$$f_{jk} = \sigma(W^{(\mathrm{f})} x_j + U^{(\mathrm{f})} h_k + b^{(\mathrm{f})}) \tag{5.5}$$

$$o_j = \sigma(W^{(\mathrm{o})} x_j + U^{(\mathrm{o})} \widetilde{h_j} + b^{(\mathrm{o})}) \tag{5.6}$$

$$u_j = \sigma(W^{(\mathrm{u})} x_j + U^{(\mathrm{u})} \widetilde{h_j} + b^{(\mathrm{u})}) \tag{5.7}$$

$$c_j = i_j \odot u_j + \sum_{k \epsilon C(j)} f_{jk} \odot c_k \tag{5.8}$$

$$h_j = o_j \odot tanh(c_j) \tag{5.9}$$

The Tree-LSTM learns a question by passing the sequence of words and the tree structure. Although the tree begins at the root, the model recursively traverses the tree and first learns the hidden states of the leaf nodes. The state of the leaf nodes are used by their corresponding parents to derive their state and so on until the network finally reaches the root node. So learning occurs breadth first from the leaf to the root. Finally, the output from the root node is converted into a N_t dimensional vector using a softmax classifier where N_t is the number of templates which in this case is 15. Formally, to predict template $\hat{t}$ from the set of templates T we calculate the softmax at the root node followed by the argmax to classify the template for the given question as shown below:

$$\begin{aligned} \hat{p}_\theta(t|x_{root}) = softmax(W^{(\mathrm{s})} h_{root} + b^{(\mathrm{s})}), \\ \hat{t} = \operatorname*{argmax}_t \hat{p}_\theta(t|x_{root}) \end{aligned} \tag{5.10}$$

The cost function is the negative log-likelihood of the true class label y and λ is the L2-Regularization hyperparameter as given below:

$$J(\theta) = -log\hat{p}_\theta(y|x_{root}) + \frac{\lambda}{2}||\theta||_2^2 \tag{5.11}$$

5.4.3 Implementation

The model was implemented using the Pytorch deep learning framework. The original Tree-LSTM implementation which is based on Tai *et al.* (2015) is available here [3]. The template classification model was adapted from this source code and is available on GitHub [4].

The major advantages in using Pytorch are:

1. Pytorch's usage of a dynamic computation graph: Deep learning frameworks maintain a computational graph that defines the order of computations that are required to be performed. What typically happens in these frameworks is that, a language (in this case Python) is used to define the computation graph and an execution mechanism (C++) reads and executes the graph. This setup is motivated due to efficiency concerns. Hence, most deep learning frameworks use a static computation graph (memory and computation needs calculated in advance before the graph is executed) for better optimization and performance. But for NLP applications the input is typically of variable length and hence a dynamic computation graph is essential. Pytorch excels in this dynamic computation approach and has its built-in from the ground up while in other frameworks such as Tensorflow such behavior needs to be bootstrapped as needed which although possible can be quite cumbersome. This availability of dynamic computation as the execution mechanism was critical in choosing Pytorch for developing the system.

2. Strong developer community with vast resources to get started and easy availability of readable and documented source code or packages for most standard neural network architectures that can be easily tailored to specific problems.

[3]https://github.com/dasguptar/treelstm.pytorch

[4]https://github.com/ram-g-athreya/RNN-Question-Answering

5.5 Conclusion

The next chapter expounds how the results from the template classification task are used by the proposed system to fill the variable slots in the predicted template to generate the appropriate SPARQL query which can answer the given question.

Chapter 6

SLOT FILLING AND QUERYING

6.1 Slot Filling

For a given input question, the template classification algorithm from the previous chapter determines the top-2 templates that are most likely to answer the question. The reason for considering top-2 templates is because of misclassification between specific pairs of templates which is discussed in detail in section 7.2. So, to improve overall accuracy the top-2 templates from the model are considered which is basically the argmax and 2nd highest argmax of the softmax output from the neural network model.

This narrows the possibilities of SPARQL queries that need to be considered. The template typically captures the semantic structure of the user's query which is then mapped to the underlying knowledge graph leaving gaps only for the slots that need to be injected as needed. The candidate SPARQL template broadly contains three kinds of slots that need to be filled:

1. **Resources:** Resources are named entities (proper nouns) which can be detected using standard named entity recognition tools. For example, London, Microsoft, etc.

2. **Predicates:** Predicates are nouns, adjectives, or verbs that may modify a resource. For example, born, capital, etc.

3. **Ontology Classes:** Ontology classes which are associated with resource define the type of class a resource might fall under. For example, when considering

the resource Barack Obama (http://dbpedia.org/page/Barack_Obama) a valid ontology class would be Person (http://dbpedia.org/ontology/Person). The ontology classes are linked through the *rdf:type* (https://www.w3.org/1999/02/22-rdf-syntax-ns#type) predicate of the target resource.

For example, consider the question *"Philadelphia City Council is the governing body of which city ?"*. The underlying candidate template detected for this question from the template classification algorithm would be Template 1:

```
SELECT DISTINCT ?uri WHERE { ?uri <p ><r >.
        OPTIONAL { ?uri rdf:type <class >}
  }
```

As can be seen, for answering this question one resource, one predicate and one ontology class needs to be detected. It must be noted that the ontology class detection is optional and even though the original candidate SPARQL query from the LC-QuAD dataset does not require an ontology class the present system requires it since templates 1 and 301 were merged during the data preprocessing step.

Since specific components/libraries are optimized for named entity recognition and relation extraction tasks an ensemble of tools were used for the slot filling process. The reason for using multiple tools for a given task was to cover the weaknesses of each while at the same time maximizing their strengths so as to produce the best possible results.

For named entity recognition DBpedia Spotlight and TagMe were used in conjunction. Mendes *et al.* (2011) developed DBpedia Spotlight for automatically annotating text documents with DBpedia URIs aka resources. DBpedia Spotlight allows users to configure the annotations to their specific needs through the DBpedia Ontology and quality measures such as prominence, topical pertinence, contextual ambiguity and disambiguation confidence. For the slot filling task a confidence of 0.4 (which is the default confidence setting in DBpedia Spotlight) was used while attempting to detect named entities in the input question. Ferragina and Scaiella (2010) implemented Tagme a system that is able to efficiently and judiciously augment a plain-text with pertinent hyperlinks to Wikipedia pages. The specialty of Tagme is that it may annotate texts which are short and poorly composed (which makes it ideal for question answering) to underlying Wikipedia pages and their inter-relations. Singh *et al.* (2018) showed in their analysis that Tagme outperforms other Named Entity Recognition tools on the LC-QuAD dataset and hence it was a natural choice for this task. But Tagme suffers when it comes to detection of single word entities such as Geneva (http://dbpedia.org/resource/Geneva) in the question: *"Is Esther Alder the mayor of Geneva?"*. But DBpedia Spotlight has a higher accuracy in spotting such short entities while it suffers against multi-word entities which are detected more efficiently by Tagme hence making them complimentary solutions for the resource detection task. Hence, Tagme's results were augmented with DBpedia Spotlight's results.

For relation linking, Singh *et al.* (2018) prescribes that RNLIWOD [1] has the best overall performance on the LC-QuAD dataset but their results show that RNLIWOD has poor overall macro performance (0.25 precision, 0.22 recall and 0.23 F-1 score). So, RNLIWOD's lexicon was augmented with a dictionary of predicates and ontology classes along with their rdfs:label (https://www.w3.org/2000/01/rdf-schema#label) used in the DBpedia Chatbot project Athreya *et al.* (2018). This resulted in higher coverage of predicates and ontology classes that could be matched with the input question thereby leading to better performance. The lexicon is basically a key value hashmap with the keys being the various surface forms that can be used to express a particular predicate or ontology class and their value being all possible predicates and ontology classes which match the sequence of words in the given surface form.

[1]https://github.com/dice-group/NLIWOD

6.2 Querying

After the candidates for each slot are detected candidate queries are built using the Cartesian product of the possible values in each slot. Each combination is queried against a DBpedia SPARQL endpoint to determine if they yield any results. This process continues until a viable combination is discovered which produces results against the endpoint. As Usbeck *et al.* (2015a) show, the problem of SPARQL query generation and pruning of invalid candidate queries is very computationally intensive and very little progress has been made beyond semantic analysis of the Cartesian product approach to improve efficiency and performance in this part of the question answering process.

6.3 Conclusion

The next chapter presents the results and in-depth analysis of both the template classification model as well as the overall results after slot filling with comparison to existing systems evaluated against the LC-QuAD and QALD-7 datsets.

Chapter 7

ANALYSIS AND RESULTS

7.1 Model Selection and Hyperparameter Tuning

Among the different models for input that were attempted the model that produced the best results was the one that used a combination of FastText Word Embedding concatenated with the One-Hot Vectors of the POS tag and word dependency relationship (RELS) derived from the syntactic parse of the sentence combined with the average of the One-Hot character vectors of each character in a given word. Fig 7.1 shows the accuracy across epochs for each of the model combinations that were considered and clearly shows that the FastText + POS + RELS + CHARS model outperforms all other combinations of input.

The preprocessed dataset containing 4920 questions was split into train and test datasets with a split of 80% training and 20% test data. The training dataset comprised of 3936 questions and the test dataset comprised of 984 questions. The accuracy of this model was 0.828 on the test dataset. The formula for calculating the accuracy is as follows:

$$accuracy(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} 1(\hat{y_i} = y_i) \tag{7.1}$$

where $\hat{y_i}$ is the predicted value of the ith example, y is the corresponding true value and N is the total number of examples.

Table 7.1 tabulates the hyperparameters of the model. The input vector was the concatenated 444-dimensional word vector. The optimizer used was the Adam Optimizer Kingma and Ba (2014) with a mini batch size of 25 examples. The loss

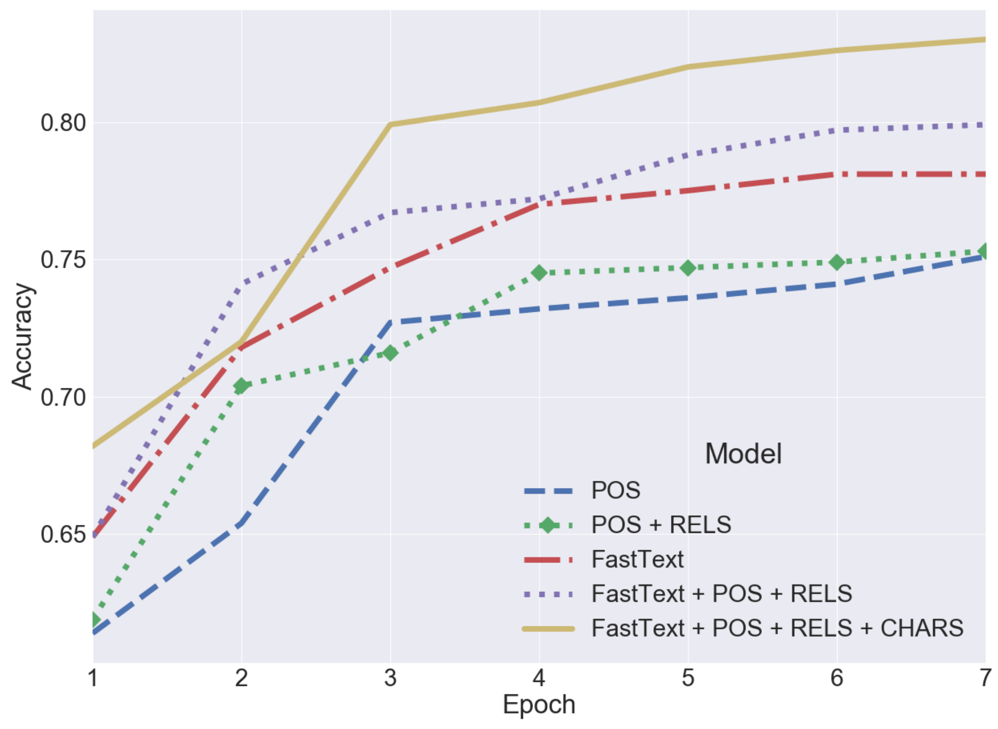


Figure 7.1: Accuracy on Test Dataset for different input models

function used was Cross - Entropy Loss which has been shown to exhibit superior performance for tasks involving multivariate classification Janocha and Czarnecki (2017).

Due to the low number of training examples the model had to be aggressively regularized and the learning rate periodically curtailed to prevent overfitting while simultaneously improving the model's generalization performance. Three strategies were employed to achieve this:

1. **Weight Decay:** Weight Decay, also called as L2 regularization is a component of the weight update rule that updates the weights after each pass by multiplying it with a factor less than 1. This prevents the weights from growing too large, and can be seen as gradient descent on a quadratic regularization term (basically

akin to L2 Normalization) Krogh and Hertz (1992). The model used a weight decay of 2.25 x 10^{-3}.

2. **Dropout:** Deep neural nets with a large number of parameters are very powerful machine learning systems. Overfitting is an issue in such linear networks. The key idea behind dropout is to randomly drop units (along with their connections) from the neural network during training. This prevents the model from developing complex co-adaptations on the training data and thereby mitigating overfitting Srivastava *et al.* (2014). The model uses a dropout of 0.2.

3. **Adaptive Learning Rate:** The optimal initial learning rate for the model was found to be 1 x 10^{-2}. But in subsequent epochs it was discovered through exhaustive experimentation that the model quickly overfitted on the training dataset while performance plateaued on the test dataset. To mitigate this, the learning rate was reduced by a constant factor periodically after a predefined number of epochs. In the model a step decay factor of 0.25 was implemented to reduce the learning rate once every 2 epochs to prevent the model from overfitting on the dataset.

Table 7.1: Model Parameters

Parameter	**Value**
Input Dimensions	444 x 1
LSTM Memory Dimensions	150 x 1
Epochs	7
Mini Batch Size	25
Learning Rate	1 x 10^{-2}
Weight Decay (Regularization)	2.25 x 10^{-3}

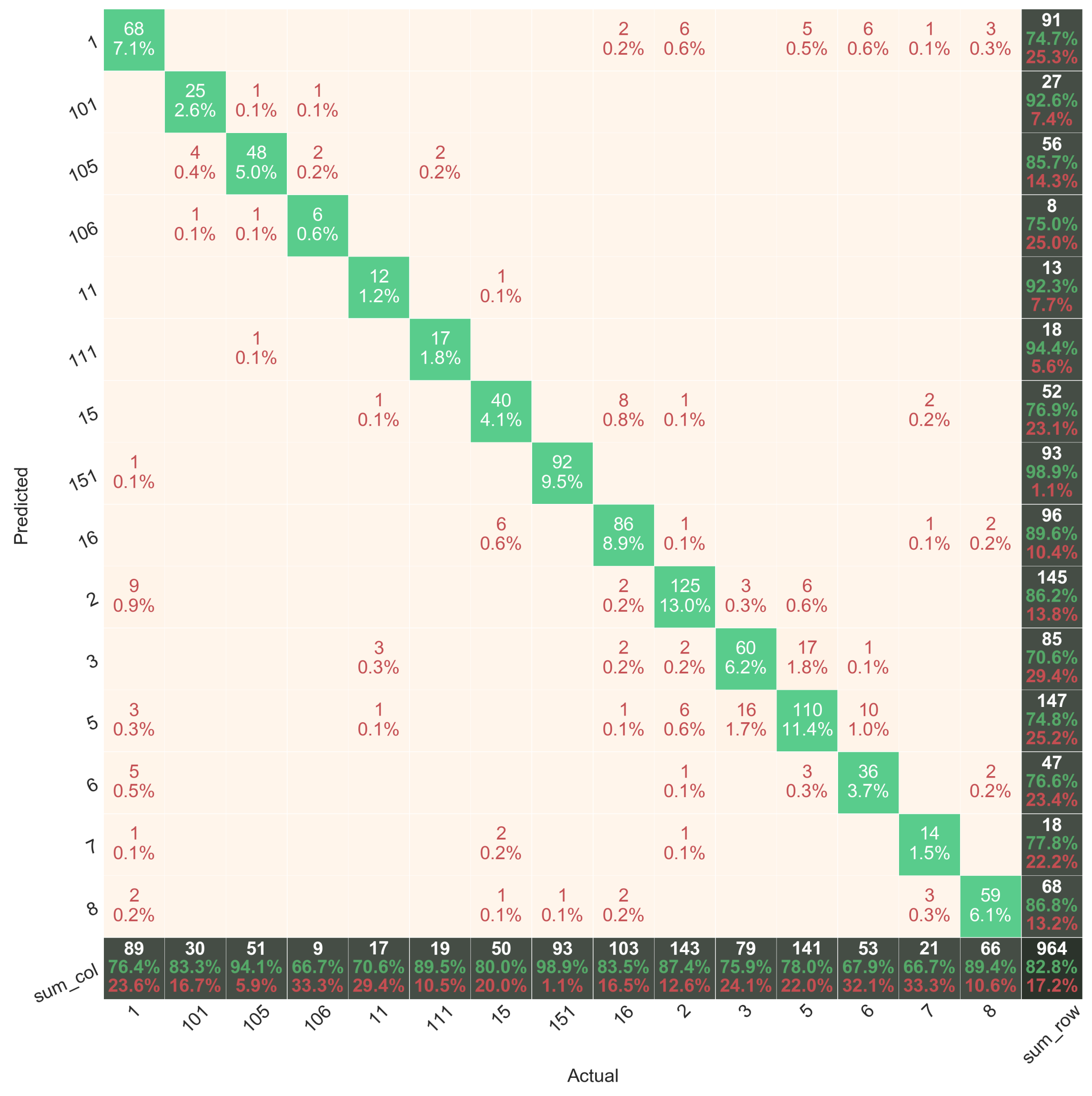
Embedded Learning Rate	1 x 10^{-2}
Dropout	0.2
Loss Function	Cross - Entropy Loss
Optimizer	Adam Optimizer
Learning Rate Scheduler	Stepwise Learning Rate Decay
Step LR Step Size	Once every 2 epochs
Step LR Decay	0.25

7.2 Template Classification

The best model from the template classification task produced an accuracy of 0.828 and 0.945 when considering the top-2 templates. Table 7.2 displays template level accuracy. The number of examples does not seem to affect the accuracy at the template level.

Rather, based on the confusion matrix from Fig 7.2 it can be observed that specific templates misclassify each other. For example, templates 3 and 5 are more likely to misclassify each other and the same can be said for 5 and 6 but 3 and 6 do not misclassify. Also template 1 misclassifies with template 2 at a much higher rate since they are basically a single triple pattern which are mirrors of each other. That is, template 1 has the triple pattern **?uri** <**p** ><**r** > while template 2 has the triple pattern <**r** ><**p** >**?uri**.

To understand this phenomenon further, the softmax outputs of each question was aggregated at the template level and the centroid was calculated by combining the word vectors for each template. Then, the Pearson's correlation was calculated for each pair of templates and a heatmap was generated considering only values 0.5 to 1 with the rest being white. The contours of the heatmap represented in Fig 7.3 is


Predicted
Actual

	1	101	105	106	11	111	15	151	16	2	3	5	6	7	8	sum_row
1	68 7.1%								2 0.2%	6 0.6%		5 0.5%	6 0.6%	1 0.1%	3 0.3%	91 74.7% 25.3%
101		25 2.6%	1 0.1%	1 0.1%												27 92.6% 7.4%
105		4 0.4%	48 5.0%	2 0.2%		2 0.2%										56 85.7% 14.3%
106		1 0.1%	1 0.1%	6 0.6%												8 75.0% 25.0%
11					12 1.2%		1 0.1%									13 92.3% 7.7%
111			1 0.1%			17 1.8%										18 94.4% 5.6%
15					1 0.1%		40 4.1%		8 0.8%	1 0.1%				2 0.2%		52 76.9% 23.1%
151	1 0.1%							92 9.5%								93 98.9% 1.1%
16							6 0.6%		86 8.9%	1 0.1%				1 0.1%	2 0.2%	96 89.6% 10.4%
2	9 0.9%								2 0.2%	125 13.0%	3 0.3%	6 0.6%				145 86.2% 13.8%
3					3 0.3%				2 0.2%	2 0.2%	60 6.2%	17 1.8%	1 0.1%			85 70.6% 29.4%
5	3 0.3%				1 0.1%				1 0.1%	6 0.6%	16 1.7%	110 11.4%	10 1.0%			147 74.8% 25.2%
6	5 0.5%									1 0.1%		3 0.3%	36 3.7%		2 0.2%	47 76.6% 23.4%
7	1 0.1%						2 0.2%			1 0.1%				14 1.5%		18 77.8% 22.2%
8	2 0.2%						1 0.1%	1 0.1%	2 0.2%					3 0.3%	59 6.1%	68 86.8% 13.2%
sum_col	89 76.4% 23.6%	30 83.3% 16.7%	51 94.1% 5.9%	9 66.7% 33.3%	17 70.6% 29.4%	19 89.5% 10.5%	50 80.0% 20.0%	93 98.9% 1.1%	103 83.5% 16.5%	143 87.4% 12.6%	79 75.9% 24.1%	141 78.0% 22.0%	53 67.9% 32.1%	21 66.7% 33.3%	66 89.4% 10.6%	964 82.8% 17.2%

Figure 7.2: Confusion Matrix

very much similar to the confusion matrix from Fig 7.2. This seems to indicate that the structure of the questions for the misclassified templates are viewed the same or similarly according to the model. Looking more deeply at the actual questions from the dataset it can be gathered that the questions are actually very similar in structure and what is different is the order of the SPARQL triple pattern which is dictated by the knowledge graph and cannot be pinned down with syntactic parsing alone.

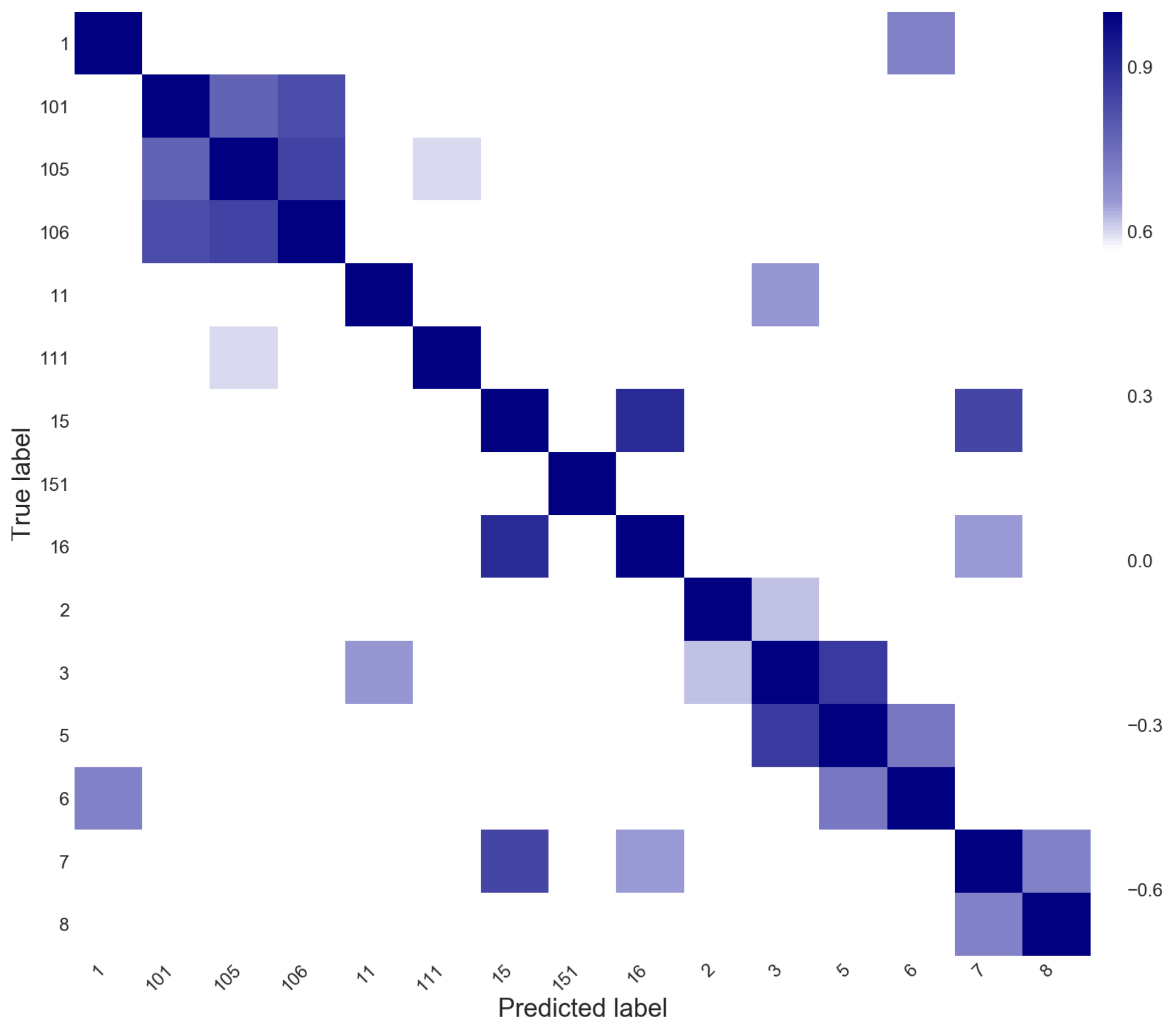


Figure 7.3: Pearson Matrix

Table 7.2: Template Level Model Accuracy on LC-QuAD dataset

Template ID	**No of Examples**	**Accuracy**
2	143	0.87
5	141	0.78
16	103	0.83
151	93	0.98

1	89	0.76
3	79	0.75
8	66	0.89
6	53	0.67
105	51	0.94
15	50	0.80
101	30	0.83
7	21	0.66
111	19	0.89
11	17	0.70
106	9	0.66

To test how well the model generalizes it was also tested on the Question Answering Over Linked Data (QALD-7) Usbeck *et al.* (2017) multilingual dataset without any additional training or optimizations. Basically, the model had never seen the dataset before and hence can serve as a good candidate to test the model's predictive power on never before seen data.

The test dataset was not considered since all 50 questions in that dataset contained multiple predicates from other schemas, predominantly Dublin Core.

The training dataset contains 215 questions of which 85 examples were eliminated and the model was tested on a total of 130 examples which is roughly 60% of the dataset and represented 7 templates which were analogous in the LC-QuAD dataset. The remaining questions were manually tagged by us based on the similarity of their SPARQL queries to the LC-QuAD dataset. The reasons why questions were eliminated are as follows:

1. **Filter based queries:** As already mentioned during the overview the LC-QuAD dataset currently does not support FILTER or OPTIONAL queries which do feature in the QALD dataset. Hence such queries had to be eliminated while testing. An example from the dataset of such a question would be: "Is Frank Herbert still alive?" which has the SPARQL query **ASK WHERE { OPTIONAL { <http://dbpedia.org/resource/Frank_Herbert > <http://dbpedia.org/ontology/deathDate >?date . } FILTER (!BOUND(?date)) }**

2. **UNION Queries:** Union queries are queries that combine two select statements. For example consider the following question from the dataset: "Give me a list of all critically endangered birds." whose corresponding SPARQL template query is **SELECT DISTINCT ?uri ?p WHERE { ?uri a <http://dbpedia.org/ontology/Bird >. { ?uri <http://dbpedia.org/ontology/conservationStatus >'CR' ^^<http://www.w3.org/2001/XMLSchema#string >. } UNION { ?uri <http://purl.org/dc/terms/subject > <http://dbpedia.org/resource/Category:Critically_endangered_animals >. } }**

3. **MinMax Queries:** MinMax queries as the name suggests are natural language questions which ask for a variation of minimum or maximum of something eg: highest, lowest, largest, smallest, longest, shortest, etc. An example query of this kind would be: "What is the highest mountain in Australia?". Currently such templates are not available in LC-QuAD to train the model.

4. **Many Triples:** Some questions in the dataset require queries of 3 or more triples to answer.

5. **Complex Boolean Questions:** Currently LC-QuAD's Boolean questions have only a single triple in the question body. In contrast, the QALD dataset also contains examples of questions with 2 triples and several variations of complex queries for boolean questions which LC-QuAD does not support.

6. **Template not available in LC-QuAD:** For example consider the following question: "Who is the king of the Netherlands ?". The SPARQL template for this question is **SELECT DISTINCT ?uri WHERE { <r ><p >?uri . ?uri a <class >. }** which is currently not supported by LC-QuAD.

The overall accuracy was 0.618 and the top-2 accuracy was 0.786. It is clear that the performance considerably varies per dataset. This is because the quality of questions differs across datasets. Quality has various dimensions, such as complexity or expressiveness. Table 7.3 shows the template level breakdown for accuracy in the QALD dataset. As can be seen template 2 is over represented compared to other templates with some templates such as template 5 and template 11 having only 1 example. But the top 3 templates (by number of examples) which comprises 84% of the dataset have competent top-2 accuracy which shows reasonable generalization power for the template classification model.

Table 7.3: Template Level Model Accuracy on QALD dataset

Template ID	**No of Examples**	**Accuracy**	**Top-2 Accuracy**
2	80	0.68	0.84
1	18	0.66	0.94
151	12	1.0	1.0
3	12	0.25	0.42

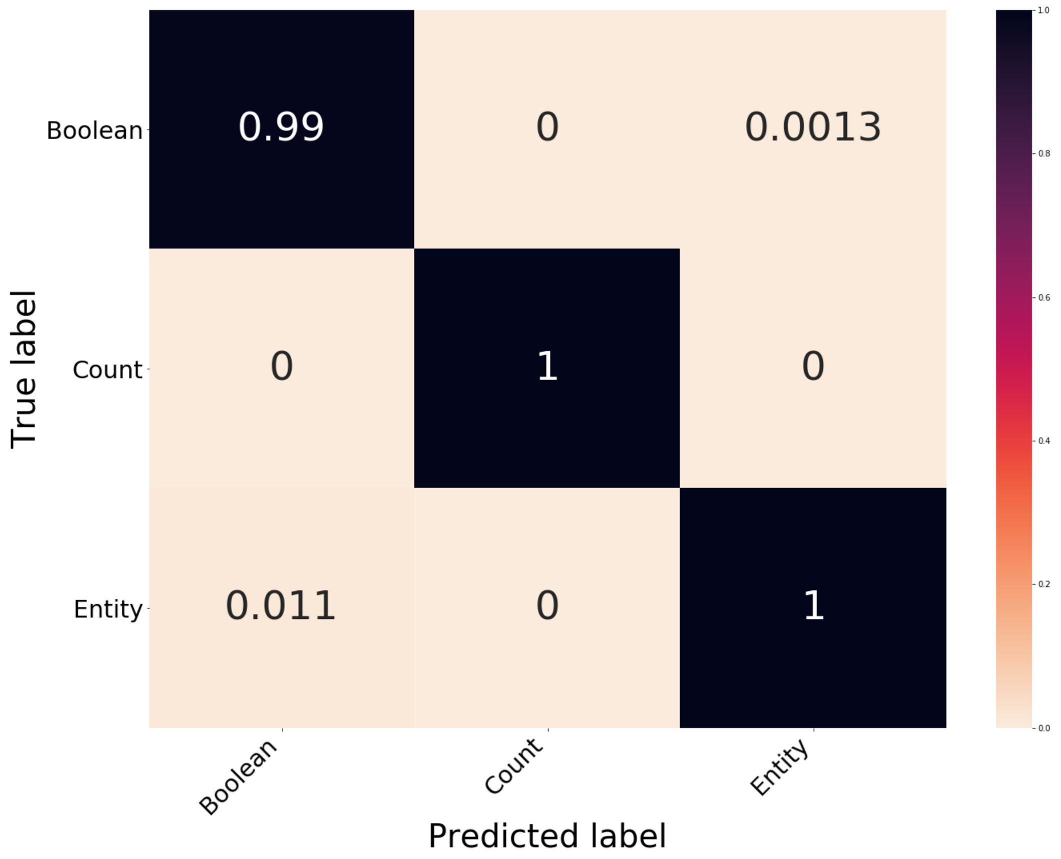


Figure 7.4: Answer Type Detection

8	6	0.00	0.33
5	1	0.00	0.00
11	1	0.00	0.00

An interesting byproduct of the model when grouped together based on the type of question i.e. entity, count or boolean without the need for modeling question word or answer type detection.

7.3 Slot Filling

Usbeck *et al.* (2015b) proposed GERBIL, a general purpose evaluation framework for bench-marking different question answering systems. The advantage of GERBIL is that it provide developers, end users and researchers with easy-to-use interfaces that allow for the agile, fine-grained and uniform evaluation of annotation tools on multiple datasets. In particular, GERBIL provides comparable results to tool developers so as to allow them to easily discover the strengths and weaknesses of their implementations with respect to the state of the art. The system ensures reproducibility and archiving of results and generates data in a machine readable format, allowing for the efficient querying and post-processing of evaluation results. The GERBIL system was used to evaluate the proposed system both on the LC-QuAD dataset as well as on the QALD-7 dataset. Table 7.4 shows the performance of the system on the LC-QuAD test dataset and Table 7.5 shows the performance of the system on QALD-7 train dataset along with a comparison of the latest question answering systems bench-marked on that dataset. A brief description on each of the metrics used in the evaluation is given below:

- **Micro Precision:** The ratio of correct answers vs total number of answers retrieved.
- **Micro Recall:** The ratio of correct answers retrieved vs gold standard answers.
- **Micro F-Score:** : The harmonic mean of micro precision and micro recall.
- **Macro Precision:** The average of the Micro Precision over all questions.
- **Macro Recall:** The average of Micro Recall over all questions.
- **Macro F-Score:** The harmonic mean of Macro Precision and Macro Recall.

Table 7.4: Performance of system on LC-QuAD

LC-QuAD Test	**Resource**	**Predicate**	**Ontology Class**	**Overall**
Micro Precision	0.802	0.950	0.976	0.135
Micro Recall	0.150	0.178	0.206	0.064
Micro F-1 Measure	0.253	0.300	0.341	0.087
Macro Precision	0.218	0.266	0.271	0.416
Macro Recall	0.215	0.258	0.261	0.428
Macro F-1 Measure	0.216	0.260	0.264	0.419

Table 7.5: Performance comparison on QALD-7

QALD-7 Train	**WDAqua**	**ganswer2**	**Proposed System**
Micro Precision	-	0.113	0.757
Micro Recall	-	0.561	0.466
Micro F-1 Measure	-	0.189	0.577
Macro Precision	0.490	0.557	0.416
Macro Recall	0.54	0.592	0.423
Macro F-1 Measure	0.510	0.556	0.417

Some of the reasons for errors in the named entity recognition task were:

1. **Specific instance detection:** Sometimes a specific form of an entity gets detected instead of the generic variety. For example, considering the question: *"How many schools have bison as a mascot ?"* the entity American Bison (http://dbpedia.org/resource/American_bison) was annotated instead of the generic bison (http://dbpedia.org/resource/Bison).

2. **Disambiguation:** Sometimes it was hard to figure out the right entity to map to the resource when there were partial matches between the sequence of words in the question and the label of the corresponding entity. For example, consider the question: *"Was 2658 Gingerich discovered in Harvard ?"*. Even though Harvard University (http://dbpedia.org/resource/Harvard_University) has a higher PageRank in the DBpedia knowledge graph and would be the correct choice for most questions in this particular case the correct entity is Harvard College (http://dbpedia.org/resource/Harvard_College). But, as can be seen, based on the question alone it is reasonable to consider either entity to be a possible fit for the given question.

3. **Accented (Unicode) Characters:** Entities with accented or foreign characters were detected poorly by both entity recognition tools. eg: Étienne Biéler (http://dbpedia.org/resource/Étienne_Biéler).

4. **Colloquialisms:** Colloquial forms referring to well known entities were hard to detect. For example, when considering the question *"How many companies were started in the states ?"* the phrase "the states" refers to USA (http://dbpedia.org/resource/United_States) but instead State (Political) (http://dbpedia.org/resource/State_(polity)) was detected.

Some of the reasons for errors in the relation extraction task (predicate and ontology class detection) were:

1. **Implicit Predicates:** Sometimes the predicate needed to answer the question cannot be inferred from the question. For example, consider the question *"How many golf players are there in Arizona State Sun Devils ?"* its corresponding SPARQL query (Template 101) is:

   SELECT DISTINCT COUNT(?uri) WHERE {

?uri <http://dbpedia.org/ontology/college > <http://dbpedia.org/resource/Arizona_State_Sun_Devils >. ?uri <http://www.w3.org/1999/02/22-rdf-syntax-ns#type > <http://dbpedia.org/ontology/GolfPlayer >} . To answer the question the predicate college (http://dbpedia.org/ontology/college) needs to be detected but this is impossible to do so with existing methods based on just the input question alone.

2. **Abbreviations:** Some questions used abbreviations instead of their expanded form which relation linking tools struggled to detect. Eg: PM for Prime Minister (http://dbpedia.org/ontology/primeMinister).

3. **Disambiguation:** The same issue of disambiguation also plagues relation linking as it did for named entity recognition. For example, consider the question: *"What is the label of Double Diamond (album) ?"* refers to record label (http://dbpedia.org/ontology/recordLabel) which was difficult for the system to detect.

4. **Subset predicates:** Sometimes specific forms of a predicate needed to be detected e.g., head coach (http://dbpedia.org/property/headCoach) instead of coach (http://dbpedia.org/property/coach).

Chapter 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

This thesis presents a novel approach for question answering over Linked Data task by converting it into a template classification task followed by a slot filling task. Although earlier approaches, most notably Unger *et al.* (2012) have attempted a similar approach this was probably the first time that neural networks, specifically recursive neural networks were applied for the template classification task. For completeness, a slot filling approach using an ensemble of the best components for the named entity recognition and relation linking tasks were presented. The slot filler utilizes the Cartesian product of detected entities for each possible slot to derive candidate SPARQL queries which are then queried against the knowledge graph to discover the best possible answer.

The key contributions of this thesis are:

1. State-of-the-art neural network techniques such as LSTM, recursive neural networks and word embeddings can be used to build a model that can address the template classification task. Using such a model removes the need for feature engineering. The template classification model achieved an accuracy of 0.828 accuracy and 0.945 top-2 accuracy on the LC-QuAD dataset and an accuracy of 0.6183 and 0.786 top-2 accuracy on the QALD-7 dataset.

2. Since the template classification model was developed without any domain specific information/features it can easily be transferred across domains.

3. An interesting finding from the template classification model was the pairwise misclassification of specific templates due to similarities in question structure but variance in the underlying provenance of the knowledge graph. This insight points to a potential future research direction on how to reconcile natural language structures to their corresponding knowledge graph topology.

4. Another interesting finding from the thesis would be a possible application of the model for answer type detection. Presently, many systems especially question answering pipelines such as Singh *et al.* (2018) have a dedicated answer type detection module to funnel queries to different sub-component question answering systems that are best suited for answering questions of a specific answer type. For such systems the present recursive neural network model can serve as a drop-in answer type detection module since it requires no new additional feature engineering (beyond the words and their parse tree) such as WH-determiner (who, what, where, when, etc.) detection, length of the word, etc.

5. After slot filling the system achieves a macro F-score 0.419 on the LC-QuAD dataset and a macro F-score of 0.417 on the QALD-7 dataset.

8.2 Future Work

Although LC-QuAD provides 38 unique templates many of them do not have adequate examples for training neural network models. Also, the coverage of templates are not adequate for addressing all possible question types. The most important types being:

1. OPTIONAL queries

2. UNION Queries

3. FILTER and LIMIT queries
4. MINMAX queries
5. Queries involving predicates from multiple schema eg: foaf, Dublin Core, etc.

Finding methods and datasets for training such questions to improve quantity and quality of data would be an obvious future direction for this work. Efforts also need to be made to integrate information from multiple sources especially Wikidata to construct more robust question answering systems in the future.

Since acquiring such data is hard the template classification approach can be extended. Instead of having a single model that classifies the entire SPARQL query, the dataset can be modified in such a way that models can be developed to predict segments of the final SPARQL query that would be needed to answer the question. For example, a model could predict the answer type (entities, count, boolean) while another tries to determine the structure of the WHERE condition (single triple or multiple triples) and yet another determines if an additional triple with *rdf:type* is needed. Then the results of all three can be combined to generate the candidate template. Theoretically, such an approach should be able to give good results even with the current paucity of data.

As mentioned earlier, a major strength of this approach is that it is domain independent. So an ideal offshoot of this research would be the application of this approach to data rich domains such as biological or biomedical data which can add to the veracity of the approach and hopefully produce interesting results.

For the slot filling task a simple algorithm using Cartesian Product was applied. Instead, application of transfer learning could be explored where information from the previous template classification task such as SPARQL query structure could be used to intelligently identify entities that fit specific sub-graphs within the template.

Using this approach has the potential of drastically improving querying performance and reducing the number of candidate queries to be considered. Also, relation extraction continues to be the Achilles' Heel of questioning answering in linked data. An interesting future direction would be to use the work being done in word embedding being applied to the relation extraction problem to see if promising results can be discovered.

REFERENCES

Athreya, R. G., A.-C. Ngonga Ngomo and R. Usbeck, "Enhancing community interactions with data-driven chatbots–the dbpedia chatbot", in "Companion of the The Web Conference 2018 on The Web Conference 2018", pp. 143–146 (International World Wide Web Conferences Steering Committee, 2018).

Bizer, C., T. Heath and T. Berners-Lee, "Linked data: The story so far", in "Semantic services, interoperability and web applications: emerging concepts", pp. 205–227 (IGI Global, 2011).

Bojanowski, P., E. Grave, A. Joulin and T. Mikolov, "Enriching word vectors with subword information", arXiv preprint arXiv:1607.04606 (2016).

Bordes, A., N. Usunier, S. Chopra and J. Weston, "Large-scale simple question answering with memory networks", arXiv preprint arXiv:1506.02075 (2015).

Cai, Q. and A. Yates, "Large-scale semantic parsing via schema matching and lexicon extension", in "Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)", vol. 1, pp. 423–433 (2013).

Chen, D. and C. Manning, "A fast and accurate dependency parser using neural networks", in "Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)", pp. 740–750 (2014).

Chinea, A., "Understanding the principles of recursive neural networks: A generative approach to tackle model complexity", in "International Conference on Artificial Neural Networks", pp. 952–963 (Springer, 2009).

Diefenbach, D., V. Lopez, K. Singh and P. Maret, "Core techniques of question answering systems over knowledge bases: a survey", Knowledge and Information systems pp. 1–41 (2017a).

Diefenbach, D., K. Singh, A. Both, D. Cherix, C. Lange and S. Auer, "The qanary ecosystem: getting new insights by composing question answering pipelines", in "International Conference on Web Engineering", pp. 171–189 (Springer, 2017b).

Dubey, M., D. Banerjee, D. Chaudhuri and J. Lehmann, "Earl: Joint entity and relation linking for question answering over knowledge graphs", arXiv preprint arXiv:1801.03825 (2018).

Elman, J. L., "Finding structure in time", Cognitive science **14**, 2, 179–211 (1990).

Ferragina, P. and U. Scaiella, "Tagme: on-the-fly annotation of short text fragments (by wikipedia entities)", in "Proceedings of the 19th ACM international conference on Information and knowledge management", pp. 1625–1628 (ACM, 2010).

Hochreiter, S., "The vanishing gradient problem during learning recurrent neural nets and problem solutions", International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6**, 02, 107–116 (1998).

Hochreiter, S. and J. Schmidhuber, “Long short-term memory”, Neural computation **9**, 8, 1735–1780 (1997).

Höffner, K., S. Walter, E. Marx, R. Usbeck, J. Lehmann and A.-C. Ngonga Ngomo, “Survey on challenges of question answering in the semantic web”, Semantic Web **8**, 6, 895–920 (2017).

Janocha, K. and W. M. Czarnecki, “On loss functions for deep neural networks in classification”, arXiv preprint arXiv:1702.05659 (2017).

Kaufmann, E. and A. Bernstein, “How useful are natural language interfaces to the semantic web for casual end-users?”, in “The Semantic Web”, pp. 281–294 (Springer, 2007).

Kingma, D. P. and J. Ba, “Adam: A method for stochastic optimization”, arXiv preprint arXiv:1412.6980 (2014).

Krogh, A. and J. A. Hertz, “A simple weight decay can improve generalization”, in “Advances in neural information processing systems”, pp. 950–957 (1992).

Lehmann, J., R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer *et al.*, “Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia”, Semantic Web **6**, 2, 167–195 (2015).

Li, Y. and T. Yang, “Word embedding for understanding natural language: A survey”, in “Guide to Big Data Applications”, pp. 83–104 (Springer, 2018).

Lopez, V., V. Uren, M. Sabou and E. Motta, “Is question answering fit for the semantic web?: a survey”, Semantic Web **2**, 2, 125–155 (2011).

Marcus, M. P., M. A. Marcinkiewicz and B. Santorini, “Building a large annotated corpus of english: The penn treebank”, Computational linguistics **19**, 2, 313–330 (1993).

McGuinness, D. L., “Question answering on the semantic web”, IEEE Intelligent Systems **19**, 1, 82–85 (2004).

Mendes, P. N., M. Jakob, A. García-Silva and C. Bizer, “Dbpedia spotlight: shedding light on the web of documents”, in “Proceedings of the 7th international conference on semantic systems”, pp. 1–8 (ACM, 2011).

Mikolov, T., “Statistical language models based on neural networks”, Presentation at Google, Mountain View, 2nd April (2012).

Saleem, M., S. N. Dastjerdi, R. Usbeck and A.-C. N. Ngomo, “Question answering over linked data: What is difficult to answer? what affects the f scores”, in “Joint Proceedings of BLINK2017: 2nd International Workshop on Benchmarking Linked Data and NLIWoD3: Natural Language Interfaces for the Web of Data co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21st-to-22nd”, (2017).

Singh, K., A. S. Radhakrishna, A. Both, S. Shekarpour, I. Lytra, R. Usbeck, A. Vyas, A. Khikmatullaev, D. Punjani, C. Lange *et al.*, "Why reinvent the wheel: Let's build question answering systems together", in "Proceedings of the 2018 World Wide Web Conference on World Wide Web", pp. 1247–1256 (International World Wide Web Conferences Steering Committee, 2018).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", The Journal of Machine Learning Research **15**, 1, 1929–1958 (2014).

Tai, K. S., R. Socher and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks", arXiv preprint arXiv:1503.00075 (2015).

Toutanova, K., D. Klein, C. D. Manning and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network", in "Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1", pp. 173–180 (Association for Computational Linguistics, 2003a).

Toutanova, K., D. Klein, C. D. Manning and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network", in "Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1", pp. 173–180 (Association for Computational Linguistics, 2003b).

Trivedi, P., G. Maheshwari, M. Dubey and J. Lehmann, "Lc-quad: A corpus for complex question answering over knowledge graphs", in "International Semantic Web Conference", pp. 210–218 (Springer, 2017).

Unger, C., L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber and P. Cimiano, "Template-based question answering over rdf data", in "Proceedings of the 21st international conference on World Wide Web", pp. 639–648 (ACM, 2012).

Unger, C., A.-C. N. Ngomo and E. Cabrio, "6th open challenge on question answering over linked data (qald-6)", in "Semantic Web Evaluation Challenge", pp. 171–177 (Springer, 2016).

Usbeck, R., A.-C. N. Ngomo, L. Bühmann and C. Unger, "Hawk–hybrid question answering using linked data", in "European Semantic Web Conference", pp. 353–368 (Springer, 2015a).

Usbeck, R., A.-C. N. Ngomo, B. Haarmann, A. Krithara, M. Röder and G. Napolitano, "7th open challenge on question answering over linked data (qald-7)", in "Semantic Web Evaluation Challenge", pp. 59–69 (Springer, 2017).

Usbeck, R., M. Röder, A.-C. Ngonga Ngomo, C. Baron, A. Both, M. Brümmer, D. Ceccarelli, M. Cornolti, D. Cherix, B. Eickmann *et al.*, "Gerbil: general entity annotator benchmarking framework", in "Proceedings of the 24th International Conference on World Wide Web", pp. 1133–1143 (International World Wide Web Conferences Steering Committee, 2015b).

Zaremba, W. and I. Sutskever, “Learning to execute”, arXiv preprint arXiv:1410.4615 (2014).