Insert here your thesis' task.

Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Computer Science

Master's thesis

Enhancing Question Answering with Structured Database Queries

Bc. Jan Pichl

Supervisor: Mgr. Petr Baudiš

26th May 2016

Acknowledgements

Foremost, I would like to express my deepest gratitude to my supervisor Mgr. Petr Baudiš for his support, knowledge, motivation and never-ending optimism. I would also like to thank to eClub Prague, especially to Ing. Jan Šedivý CSc., for providing the opportunity to work on this thesis and related projects and for creating the environment where I can work on the thesis in company of the other students. I am also very grateful to ČVUT Media Lab for funding this project.

Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures". Without these resources, all the experiments would be almost impossible to be done.

Last but not least, I would like to thank to my family and friends for their support and motivation.

Declaration

In Prague on 26th May 2016 . . . .

Citation of this thesis

Pichl, Jan. *Enhancing Question Answering with Structured Database Queries*. Master's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, 2016.

Abstrakt

Cílem této práce je vytvoření systému, který je schopen poskytnout odpovědi na otázky zadané v přirozeném jazyce. Jelikož otázky týkající se konkrétních faktů mají obvykle jednoduchou strukturu obsahující jednu nebo dvě entity a několik relací mezi nimi a odpovědí, nabízí se použití strukturovaných databází jako možný zdroj odpovědí. Relace, které jsou stavebním kamenem pro dotazy do těchto databází, jsou vybrány z relací nacházejících se v okolí entit obsažených v otázce. Několik typů skórování a řazení těchto relací nebo sekvencí relací je navrženo a porovnáno v této práci. Finální implementace bude součástí YodaQA systému a výsledky původního řešení budou provnány s novým přístupem.

Klíčová slova Odpovídání na otázky, Strukturovaná databáze znalostí, Zpracování přirozeného jazyka, YodaQA, Freebase, Neuronové sítě

Abstract

The aim of this thesis is to create a system capable of providing the answers to a factoid question in the natural language. Since the factoid questions usually have a simple structure consisting of one or two entities and a few relations between them and between the entity and the answer, it is suitable to use a structured knowledge base as an answer source. The relations forming a query for the knowledge base will be selected from the relation neighbourhood of the entities contained in the question. Several types of ranking those relations or sequences of the relations are introduced and compared. The final implementation will be a part of the YodaQA system and the original and the final performance will be compared.

Keywords Question Answering, Structured Knowledge Base, Natural Language Processing, YodaQA, Freebase, Neural Networks

Contents

Introduction **1**

Goals **3**

1 Question answering system **5**
1.1 Types of the questions . . . 5
1.2 IBM Watson . . . 6
1.3 YodaQA . . . 6
1.3.1 Question analysis . . . 7
1.3.2 Answer producing . . . 8
1.3.3 Answer analysis . . . 8
1.3.4 Answer scoring . . . 9

2 Analysis and Related work **11**
2.1 Knowledge bases . . . 11
2.1.1 Graph databases . . . 11
2.1.2 DBpedia . . . 14
2.1.3 Freebase . . . 14
2.1.4 Wikidata . . . 14
2.2 Problem specification . . . 15
2.3 Related work . . . 15
2.3.1 More Accurate Question Answering on Freebase . . . 15
2.3.2 Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base . . . 17
2.3.3 Enhancing Freebase Question Answering Using Textual Evidence . 19
2.3.4 Results comparison . . . 21
2.4 YodaQA approach . . . 21

3 Implementation **23**
3.1 Overview . . . 23

3.1.1 Dataset 24
3.1.2 Entity linking 25
3.1.3 Neighborhood exploring 26
3.2 Generating gold standard 28
3.2.1 Dataset for dot product based scoring 29
3.2.2 Dataset for neural network based scoring 30
3.3 Extending original YodaQA approach 31
3.4 Generating candidates 31
3.5 Dot product based selection 33
3.6 Neural network based selection 34
3.6.1 Convolutional neural network 35
3.6.2 Recurrent neural network 35
3.6.3 Other models 35
3.7 Final answer producing 36
3.7.1 Scoring API 36
3.7.2 Answer scoring and merging 37

4 Experiments **39**
4.1 Dataset experiments 39
4.1.1 Empty gold standard of the paths 39
4.1.2 Gold standard reduction 41
4.1.3 Property blacklist 41
4.2 Logistic regression 42
4.3 Dot product based scoring 44
4.4 Neural network based scoring 45

Conclusion **49**

Bibliography **51**

A Acronyms **55**

B Property blacklist **57**

C Contents of enclosed CD **59**

List of Figures

2.1 Basic RDF statement 12
2.2 Modified RDF statement 12
2.3 Simple query 13
2.4 WHERE statement with CVT node 13

3.1 Pipeline structure 24
3.2 Three basic query structures 27
3.3 Querying property label 30
3.4 Querying second level property 32
3.5 Non-siamese convolutional neural network 36
3.6 Non-siamese recurrent neural network 37

4.1 Learning curves for the dot product based model 45
4.2 F_1 score dependence on the answer score threshold 47

List of Tables

2.1 Aqqu results 17
2.2 Stagg results 18
2.3 DeepQA results 20
2.4 Results comparison 21
2.5 YodaQA results 21

4.1 Empty gold standard paths 40
4.2 Blacklisted properties 42
4.3 Original approach 42
4.4 Original approach + entity linking 43
4.5 moviesD with entity linking, movies question excluded 44
4.6 Extended approach 44
4.7 Dot product based scoring 45
4.8 Model comparison using the WQME and WQMER datasets 46
4.9 Neural network based scoring 46
4.10 Relation matching comparison 46
4.11 F_1 comparison 47

B.1 Blacklisted properties 57

Introduction

Searching for information is a very common and important task. The search engines such as Google, Bing, etc. can be used for obtaining articles containing information relevant to the given search query. People typically type a question or selected keywords and a search engine returns a list of the articles sorted by their relevance. This is convenient when someone looks for the complex information. For example, if users want all the information about one particular actor, they want to read whole article.

On the other hand, sometimes users want to know one specific fact, such as the age of an actor. In this case, they need to go through the articles returned by the search engine a look the fact up. Sometimes, the requested fact could be shown in an article snippet. With a growing number of pages in the the world wide web, the process of finding exact information in a list of the articles could become difficult task.

Question answering (QA) system, on the other hand, returns in ideal case only the exact information which is an answer to the given question. Some search engines provide this feature in addition to the search results for some subset of questions. The typical search engine query consists of only keywords as people usually do not want to type full question, whereas question answering system expects full question as an input. This makes the QA system more suitable for cases where the voice is used as an input.

The question answering is often implemented as a part of the smart assistants. The examples of such assistants are Microsoft Cortana, Apple Siri and Google Now. These services provide additional functionalities besides pure question answering such as adding reminders, sending messages, etc. Since the input is usually a user voice, the natural language sentences need to be processed for every task the same way as they need to be processed during the question answering task.

Goals

The goal of the thesis is to enhance the YodaQA question answering performance using the structured knowledge bases. This is one the possible approaches (beside full text search in unstructured text) to produce answers for given questions. In the beginning of the thesis, it is necessary to understand the question answering process from question as an input to final answer as an output.

Next important step is to get to know current state of art solutions and implement own approach of generating answers from the structured knowledge bases.

The implemented solution is a part of the YodaQA system and it is also important to analyse current approach of the system in order to improve it.

Chapter 1

Question answering system

A question answering system is, in general, a software, which accepts question in natural language and as its output it provides an answer. In comparison with the search engines, this requires an additional step, which is finding the relevant information in the articles or in other sources of information. In comparison with the querying structured information sources, the database queries need to be in strict structured format. However, the natural language questions can be written in numerous ways which makes it difficult for computers to process them and find the right answer.

1.1 Types of the questions

The question types can be divided into several categories, based on the range of covered topic:

1. Open domain
2. Domain specific

The open domain question answering system should be able to answer a question from various topics such as history, film, science, literature, etc. On the other hand, domain specific system is specialized into one topic which makes question answering process easier because of the limited range of the questions.

The question can be also divided into the categories according to the answer types. There are multiple levels of a answer types that can be distinguished. At this point, we focus on the top level answer types which could be following:

1. Description
2. Opinion
3. Summary
4. Factoid

The questions in the categories description, opinion and summary usually have an answer consisting of more sentences, while the factoid questions can be answered using a few words. The example of the description question can be: *What does the building look like?* Opinion question: *What is the author's opinion about political situation in the article?* Summary question: *What is the book about?*

The factoid questions are more straightforward and usually can be answered unambiguously. For example: *Who is the president of the USA?* or *Who played Severus Snape in Harry Potter movie?* Unlike the description, opinion and summary questions, the factoid questions do not require full text understanding and returning a text paragraph based on a knowledge from the original text. However, the answer can be exact part of the original text or the field in database.

The factoid questions can be further divided into yes/no, list and other questions. The answers to the yes/no questions cannot be typically directly obtained as a knowledge base query. For example, in order to answer the question *Did Daniel Radcliffe star in Harry Potter movies?* one need to get all the actors who starred in Harry Potter and find whether any of their names match with Daniel Radcliffe.

The answers to the list questions, however, can be directly obtained from the knowledge base. Additionally the question answering system needs to recognize that multiple answers are requested.

1.2 IBM Watson

IBM Watson is a computer capable of the questions answering task. It was specialized into the Jeopardy! quiz questions. These are in fact the factoid questions but the structure of the questions is quite different. The questions do not typically start with the question word and may consist of multiple sentences. For example, the regular factoid question such as *Who is the president of the USA?* could have quiz-like format *Say name of the current USA president.*

In 2011, Watson won the Jeopardy! competition against the human opponents. Thanks to this event, the question answering task has become even more popular and future QA systems was inspired by Watson's architecture.

1.3 YodaQA

YodaQA is a question answering system which focuses on the open domain factoid questions. The acronym stands for "Yet anOther Deep Answering pipeline". The development began in 2014 by Petr Baudiš[1] and it has been under the development since. It is an open source project with the source codes available at `github`[1]. It also provides live demo[2] where anyone can test its capabilities.

[1]https://github.com/brmson/yodaqa
[2]http://live.ailao.eu/

The system is built in Java using the Apache Unstructured Information Management Architecture (UIMA) framework[2], which helps to store annotations generated during different parts of the pipeline. The separation also makes us to easily use multiple CPU cores for different parts of the question answering process.

The process beginning with a input question and ending with a final answer can be divided into four basic parts:

1. **Question analysis** - segmentation, lemmatization, POS (part of speech) tagging, finding concept, clues, LAT, SV, ...
2. **Answer producing** - finding passages from corpus, querying knowledge bases (predicting query properties)
3. **Answer analysis** - answer types, answer features
4. **Answer scoring** - ranking answers from all sources

The following subsections describe each phase in detail.

1.3.1 Question analysis

The first step that is needed to be done is the tokenization. It simply splits the sentence or sentences into individual words which can be consequently processed in further steps.

The next step takes individual words and makes a part of speech tag for each one of them. It identifies word categories such as noun, adjective, verb or adverb. These tags are used for example for identifying entities or clues.

The lemmatization step finds a base form of the words. For some words, it needs to take context into account in order to determine proper base form. For example, the word "bark" can be a verb with base form "to bark" (meaning: *A dog is barking.*) or it can be a noun with base form "bark" (meaning: *The tree has a thick bark.*)

The focus of the questions is identified by several rules based on the information obtained in the previous step of the question analysis. The focus is the point in the question, telling what the answer is about. The focus word is also the place where we can place the answer. For example, in the question *Who played Sheldon Cooper in The Big Bang Theory?*, the word "who" indicates that the focus is "person". It can be replaced with the answer resulting in the sentence *Jim Parsons play Sheldon Cooper in The Big Bang Theory.*

The lexical answer type (LAT) is used to determine a type of the answer. It is usually a focus with an exception when focus is a question word. In that case, it needs to be transformed to the concept word according to the predefined rules. For example: *who* → *person*, *where* → *location*, *when* → *time*, etc.

The entity linking is an important step for the thesis. The goal of this subtask is to find all entities mentioned in the question and link them to the appropriate Wikipedia topics. The entity candidates are drawn from the clues (question keywords), n-grams, fuzzy search and labels from Wikipedia links (crosswikis). The candidates are then sorted by their probability estimated using a logistic regression. Features, such as entity origin, edit distance and relatedness to the question, are the inputs to the classifier.

1.3.2 Answer producing

Answers can be produced from two different sources which are the unstructured English Wikipedia text and the structured knowledge bases.

- **The unstructured search**[1] uses articles from the English Wikipedia. The article typically have an entity label as its title and the first paragraph contains a description or a summary of the given entity. The search process itself can be divided into four categories:
  - Title-in-clue search tries to find match between the question clues and the article titles. The clues could be entities, SVs, LATs, noun phrases described in previous subsection.
  - Full-text search searches for a clue in the title and in the article text. The individual sentences are marked as the passages and sent to the passage analysis.
  - Document search is similar to previous one, but only the document text is searched and the titles are considered as potential answers.
  - Concept search is similar to the first one, but the exact match between the article title (or alias) and the question clue is required.
- **Structured search** is the main focus of this thesis and will be discussed later in more detailed way. The goal is to use a structured knowledge base (Freebase, DBpedia, Wikidata) as a source of the answers. The task is to build a query in SPARQL (SPARQL Protocol and RDF Query Language) which will be executed against the knowledge base.

1.3.3 Answer analysis

This step takes the candidate answers and generates various features, which will be used for the answer scoring. The important feature is the one that tells whether the question LAT matches the answer type. It is called type coercion. For example, in the question *Who plays Marge in The Simpsons?*, the LAT derived from the question word "who" is person and the answer type of the correct answer "Julie Kavner" is a actor which can be generalized to a person using Wordnet[3] links. Since both the question LAT and the answer type are a person, this step generates perfect type coercion.

The answer LATs are generated in several ways. Answers have quantity LAT if they contains a number. The type generated by the Named entity recognizer (NER) is used as the LAT if available. These types are: person, time, organization, location, money, percentage and date. The answers generated from the unstructured text use DBpedia ontology to generate LAT whereas the answers form the structured knowledge bases use the property name.

Another feature is telling whether there is an overlap between the question clues and the answer.

1.3.4 Answer scoring

This step takes all candidate answers and computes a score for each of them, telling the possibility of an answer being the correct one. The classification is done using gradient-boosted decision forest consisting of 200 trees.

An input for the classifier are the features listed in the previous subsection plus a few additional ones. If the same answer was generated multiples times, all its instances are merged into one and the number of these instances is used as a feature. The questions are also classified into 6 classes which are used as the features. These categories are description, entity, abbreviation, human, number and location.

For each feature, two more values are generated. The first value is the normalized value of the corresponding feature over the full answer set in a way, that the mean of the distribution of the values has a mean equal to 0 and a standard deviation equal to 1. The second one is a binary value which tells whether the corresponding feature was set or not.

Chapter 2

Analysis and Related work

This chapter analyses the possibilities of the question answering using structured knowledge bases. The first section describes a knowledge base structure and principles as well as the comparison of the different kinds of them. The very next sections describes current approaches of the structured query generation. Finally, the last section analyses current approach of YodaQA system and possibilities of improvement.

2.1 Knowledge bases

The knowledge base (KB) is a storage of the information. In computer science, it requires the data to be in a machine readable format. The idea is to be able to create a query which can be used for data retrieval. The knowledge base is usually implemented as a graph database where information is stored as nodes and edges.

2.1.1 Graph databases

The graph database is a NoSQL data store which uses the graph structure for storing information instead of the tables as in case of relational databases. This approach aims to domains where there are many relationships between the individual entries in the database. The relational databases store connections between the entries in different tables as foreign keys. The keys refer to the primary keys in another table. It means that the relationship needs to be computed at the query time which can be computationally expensive.

Graph databases consist of three main construction elements:

- **Node** - representation of the entity (analogical to the row in relation DB)
- **Property** - information about the node
- **Edge** - relation between two nodes or the node and the property

The largest subset of the graph databases are **RDF**[4] databases. The RDF acronym stands for Resource Description Framework. It is the only standardized structure among the NoSQL solutions[5]. Like in the graph databases, the data are stored in the triplets which represent the object–predicate–subject structure. For example, the statement *Suzanne Collins wrote the Hunger Games.* can be graphically represented as shown in the Figure 2.1.

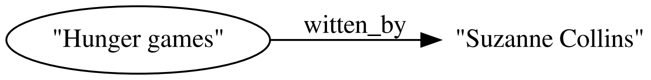


Figure 2.1: Basic RDF statement

The example sentence is basic RDF statement, where "Hunger Games" is a subject, "written_by" is a predicate and "Suzanne Colling" an object. The predicate can be also called *property name*. These property names can either connect two resources or a resource and an atomic value. In the previous example, the *Hunger Games* is the resource node and *Suzanne Collins* is the atomic value. The resources can have set of properties (predicates) leading to the other resources or atomic values, but the properties cannot lead from the atomic values. If we need to add some properties to *Suzanne Collins* we would need to modify the example in a way shown in the Figure 2.2.

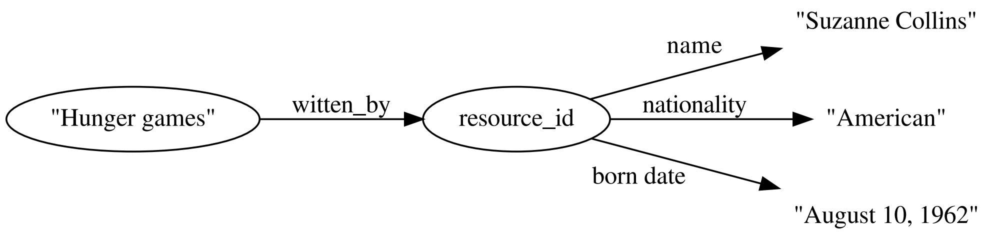


Figure 2.2: Modified RDF statement

The replacement of the atomic value with a resource node allows to add multiple properties to that node (properties *name* and *born date* were added for illustration). The resource node has now an unique ID instead of the author's name as label. Note that the node labelled "Hunger Games" should also have the unique ID instead of this label. This is for the simplicity of the example and the proper identification is shown on the other resource node.

In order to have unique identifications across knowledge bases from different sources and authors, RDF comes with XML name spaces. These name spaces point into the specific vocabularies of the property names or the node IDs. The *written_by* property name, for example, could look like *myns:written_by* if the *myns* prefix is defined, for instance, in the following way:

```
<?xml:namespace ns="http://mydomain.org/ns/" prefix="myns" ?>
```

For querying data from the RDF database, there is a strong standardized language called SPARQL (SPARQL Protocol and RDF Query Language). The syntax of this language is very similar to the SQL syntax. There are two essential keywords same as in a SQL query (`SELECT, WHERE`). Since there are no tables in RDF databases, the keyword `FROM` is not usually necessary to be present in a query. However, it can be used for specifying the RDF graph.

It is usual for the query to begin with a prefix specification. It simplifies the rest of the query because we do not have to write full name space URI. Then, it is followed by the `SELECT <variables> WHERE` { `<conditions>` }.

```
PREFIX ns: <http://rdf.freebase.com/ns/>
SELECT ?director WHERE {
  ns:m.12345 ns:film.film.directed_by ?director .
}
```

Figure 2.3: Simple query

The Figure 2.3 shows an example of a simple SPARQL query. It tries to retrieve a node connected to the node with ID `m.12345` by the relation `film.film.directed_by`. If the node with given ID was a film, the query would return its director (if it was stored in the knowledge graph). Note that the node ID and the property (relation) are prefixed with the namespace prefix, specifying the Freebase schema.

As shown in the Figure 2.3, the query structure corresponds with the RDF triplets. It means that the lines of the query also describe subject – predicate – object structure. The variable with the same names are mapped to the same object. The mapping is helpful when constructing a more complex query. For example (Figure 2.4), if we want to obtain a node that is connected to another one using two properties, we would need to pass those two lines into `WHERE` block of the query.

```
?film film.film.starring ?meta .
?meta film.performance.character ns:m.12345 .
```

Figure 2.4: `WHERE` statement with CVT node

The middle node marked as *meta* is also called Compound Value Type (CVT) and it interacts with another properties which describes the relationship between given entities (in this case, additional properties could be for instance, an actor, start date, end date, etc.).

For purpose of this thesis, one more query element is worth mentioning. We can specify a filter condition to select only subset of the results. The typical example is that we want only those nodes, which have a English label (`FILTER(LANGMATCHES (LANG(?label),"en"))`).

2.1.2 DBpedia

DBpedia[6] aims to create a structured graph from Wikipedia. Although Wikipedia stores the information mostly in an unstructured way, some important data are stored in the infoboxes. The infoboxes are tables shown in top right corner of the Wikipedia articles and they show basic information about the entity described in the article. These infoboxes also contains metadata which can be used for the automatic processing of these pieces of information and storing them into the structured knowledge base.

Thanks to the automatic process of importing data from the Wikipedia to the DBpedia, the data are updated several times per a year. The data are stored in the RDF triplets and can be queried with SPARQL. This is the main advantage in comparison with the Wikipedia, where only the full text searches can be performed.

The DBpedia ontology is created by a community as well as the mappings between the information stored in the Wikipedia infoboxes and the RDF triplets described by the ontology. The mapping solves several issues such as different names for the same properties in the infoboxes.

2.1.3 Freebase

Freebase[7] is a community based knowledge base. It was launched in 2007 by Metaweb Technologies company. The company was acquired by Google later in 2010. Its goal was to create a structured knowledge base with the information gathered from multiple sources with community help in adding new data and maintaining already stored content.

It is similar project to DBpedia with some major differences.

- The information is imported by the community instead of automatic import from wikipedia infoboxes. (It is also possible for some bots to automatically import information but it is not the only way.)
- The information comes from multiple source in addition to the Wikipedia, such as MusicBrainz or TVRage.

Thanks to the additional sources, the Freebase offers better topic coverage than the DBpedia. Unfortunately, it is no longer under development as Google decided to shut it down. The official retirement was on June 30, 2015, but the Freebase API was functional until May 1, 2016 (approximately). Company now offers Google Knowledge Graph API which can be used for searching for the entities according to the labels. There are also available Freebase RDF dumps which can be imported in a custom endpoint.

2.1.4 Wikidata

Since the end of the Freebase, Google has suggested to use Wikidata[8] as its successor. The Wikidata was launched by Wikimedia foundation in 2012. Its goal is to store the information in the structured way to help other projects including Wikipedia or the question answering systems. Using the structured information about the entities could be used for the construction of Wikipedia infoboxes. It is the exact opposite approach to DBpedia approach, where the structured information is built from the infoboxes.

The Wikidata stores the data origin or citation beside the actual information. This is the main reason why the Freebase is not directly imported into the Wikidata as the Freebase provides mostly the Wikipedia page as a source of the information according to the post on the official Freebase Google Plus page. Instead of importing whole Freebase, Google provides a mapping between the entity IDs where it is possible.

2.2 Problem specification

The problem of the answering questions using the structured knowledge bases can be described as a transformation of the natural language question to the specific logical form or the query in the specific language which can be used for retrieving the information. In the RDF knowledge bases, it means to find the correct relations (properties) which "leads" to the answer entity. It also requires to find the topic entity which is mentioned in the question where the property path should begins. More about this topic is discussed in the Chapter 3.

2.3 Related work

Next three subsections describe in detail three most recent articles about the question answering using the structured knowledge bases. All three articles (More Accurate Question Answering on Freebase[9], Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base[10] and Enhancing Freebase Question Answering Using Textual Evidence[11]) have similar approach, in which they all explore all possible property paths and rank them.

2.3.1 More Accurate Question Answering on Freebase

More Accurate Question Answering on Freebase[9] article describes the process of transforming a natural language question to a SPARQL query. The system that the article describes is called Aqqu. This process is divided into four steps: Entity identification, Template matching, Relation matching and Ranking.

Entity identification

This step tries to find the entity mentions in the input question and maps them to the nodes in the knowledge base. Even if the word or sequence of words in a sentence is correctly annotated as an entity, there could be several possibilities how to map it to the knowledge base node (e.g. people with the same name).

The words in the question are tagged by the Stanford Parser[12]. For the entity identification, tags NN (noun) and NNP (proper noun) are important. The entity candidates are generated from every subsequence of words from the question in a way that the subsequence of length one must be tagged NN and any subsequence must not split the words tagged NNP. The subsequences are then mapped to the entity titles or aliases. For this, the CrossWiki dataset[13] is used. This dataset also provides scores for the specific entity given the label. If the entity is not in the dataset, the score is computed relatively to the score of the best entity given the label.

Template matching

There are three types of the templates to be matched:

1. Single property between the topic entity node and the answer entity node
2. Two properties connected with a CVT node between the topic node and the answer node
3. Two properties connected with a CVT node between the topic node and the answer node plus one addition property from the CVT to another entity mention in the question (if possible)

All these relations can be obtained using the SPARQL. For each entity in the question, all relations which are associated with the entity node are found. If some of those relations leads to the CVT node, it is expanded same way as the entity node.

To generate third type of the template, each entity is expanded and all CVT nodes found are stored in a list along with the corresponding entity. Then the lists are intersected according to the CVT node.

Relation matching

This step computes the metrics to help to determine which query candidate generated during the template matching is the relevant one to the question.

- Literal matches - number of words from the relations that are also contained in the question
- Derivation matches - number of pairs (one token from relations, one token from question) which can be matched using WordNet derivations
- Synonym matches - number of pairs which are considered a synonym according to the distance of *word2vec* word embeddings.
- Context matches - number of precomputed indicator words for the relation connecting two entities.

Additionally, the answer type matching is done in order to check whether the answer provided by the relation has the same type which is requested in the question. This is done using the handcrafted rules which take as an input the question word and the most common types of entities into which the relation leads.

Ranking

Two types of ranking were introduced in the article: pointwise and pairwise. The pointwise ranking is a typical approach where the score is assigned to each candidate and then they are sorted accordingly. The pairwise ranking, however, estimates for each pair of the candidates, which one of them should be ranked higher. Because it does not guarantees anti-symmetry and transitivity, final ranking is done by the number of "wins" for each candidate.

The classifiers used for ranking use the features generated in the entity identification and the relation matching phases. The first ranked property path is used for retrieving final answers. The final results are demonstrated on Free917 and WebQuestions datasets. They have 267 and 2032 questions in testing split respectively. The result are shown in table 2.1.

Table 2.1: Aqqu results

Free917 (accuracy)	WebQuestions (average F_1)
65.9 %	49.4 %

2.3.2 Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base

This article[10] describes the system called STAGG which generates a query graph from the natural language question. The individual steps of the process are: Linking Topic Entity, Identifying Core Inferential Chain, Augmenting Constraints & Aggregations and Learning.

Linking Topic Entity

Since the whole process is considered as a graph construction, there is an empty graph in the beginning. The first action needed to be done is to add a node representing the topic entity. The system called S-MART[14] is used for linking entities. For each entity from the knowledge base, the list of possible forms is created using the Wikipedia redirect table and the other sources. Every subsequence of words is then used as a candidate to be matched to the entity form in a lexicon. The score is based on frequency of given entity in the lexicon.

Identifying Core Inferential Chain

The candidate graph now contains one node representing the entity. Next step is to add one or more predicates leading from the entity node. Maximum number of added predicates is two. This corresponds to the template matching step in the previous article. The only difference is that there is no "third" relation (in this step) as it was in template 3 in the previous article. There is only a direct sequence of the properties possibly leading to the node which should be the answer node. This sequence is called core inferential chain.

Augmenting Constraints & Aggregations

Given the core inferential chain containing one CVT node, it can be further enriched by additional nodes. It reduces the number of possible answers (incorrect ones) which can be accessed using this graph. One of the possible additional nodes could be another entity mentioned in the question (analogical approach to the template 3 in previous article). Another option is to add the aggregation nodes (like argmax). These nodes are added only if a certain word mapped to the node is present in the sentence. The rules for mapping words to the aggregation nodes are taken from the handcrafted list.

Learning

In the learning step, the reward function is introduced, which tells whether the generated query graph is correct or not. Several features are used for training the model:

From the entity linking step, the score of the entity is used as a feature. For measuring quality of the core inferential chain, the convolutional neural network (CNN) is used. The network accepts letter trigram vectors as an input. The trigrams are constructed from each word with additional word boundary. These vectors are projected using a convolutional layer followed by a max pooling. The final vector is an output from a feed forward layer. Two models, each containing two networks with described structure, are learned.

1. First one is used to compare the input question (with the entities replaced using the token "$< e >$") and the representation of the inferential chain (the sequence of predicates).
2. Second one is used to compare the sequence of predicates concatenated with the entity name and the input question.

Both models use two neural networks which have fixed length vector as an output. To measure the similarity, the distance function (cosine) is used.

For each constraint node, the percentage of the words from the entity occurring in the question, is used as a feature. For aggregations, the features tell whether some word from the handcrafted list appeared in the question.

Using the described features, one layer network model is used for the final ranking. The benchmark is done on the WebQuestions dataset. The results are shown in the Table 2.2.

Table 2.2: Stagg results

	Precision	Recall	F_1
WebQuestions	52.8 %	60.7 %	52.5 %

2.3.3 Enhancing Freebase Question Answering Using Textual Evidence

Enhancing Freebase Question Answering Using Textual Evidence[11] article employs some concepts used in the previous articles as well, but in general, the approach have many differences.

Question decomposition

The main difference is that the question is not processed as a whole, but it is decomposed into the subquestions instead. Then the subquestions can be answered using a single relation. This decomposition is done using the syntactic-based patterns. The individual questions are answered separately and the intersection of the answer sets is taken as the final answer set.

S-MART[14] is used for the entity linking as well as in the previous article and up to 5 top entities are used as the candidates.

Relation extraction

For every subquestion, one relation needs to be retrieved. The multi-channel convolution neural network is used for this purpose (MCCNN). As an input into this network, two types of the features are used:

1. Syntactic level features - it is the shortest path between the entity and the question word (for example the path for the question *Who did shaq first play for* is "$\leftarrow$ dobj – play – nsubj $\rightarrow$"). This path is represented as a sequence of the word vectors, dependency labels and the directions.

2. Sentence level features - it is simply the question with the entity and the question word removed.

Each type of the features is then pushed into the separate channel of the neural network. Each channel uses convolutional layer which produces a feature vector and those two vector are then concatenated. This vector is then used as an input into the softmax classifier which produces a vector with length same as the number of the predefined relations. Each field of the vector tells the probability of the corresponding relation.

Learning

For every question, a vector consisting of 1 for the correct relations and 0 for the incorrect ones is constructed as a gold standard. The predicted vector of the relations contains the corresponding probabilities. The loss function is a cross entropy between those vectors for the individual questions summed over all questions.

Using the linked entities and the extracted relations, another classifier (Support Vector Machine – SVM) is used to select the correct combination of the entity and the relation to obtain the answer. The classifier uses following features as an input.

- Score of the predicted entity
- Text overlap with the entity name
- Count of the relation phrase occurrences in the entity description (relation phrase is extracted as a syntactic level feature)
- Score of the relation
- Sum of the tf-idf metric of the question words. The documents are created for each relation separately, where each document consists of the questions from the training dataset, which have the relation in the entity neighbourhood.
- Answer type - whether the question word matches the answer entity type
- Occurrences of the last fragment of the relation in the question

The scores for the training dataset is done in the following way. All predicted entities and relations are paired with the corresponding question. If both entity and relation are correct, they are assigned with score 3. If only one them is correct, score 2 is assigned. If both are incorrect, score 1 is assigned.

Refinement

A refinement step tries to reduces the candidate answers generated from the relations obtained in the previous steps. For each topic entity, the article regarding this entity is found. Each sentence containing any of the candidate answers is stored into the list. The words form the question and from the evidence sentences are then paired. The occurrences of those pairs are used as a feature for the refinement model. The answer generated from the relations and the gold standard answers are used for the training.

The evaluation is done using the WebQuestions dataset. The results are separated according to the part of the system which were used: J - joint inference (selection of the correct combination of entity and property), R - refinement. The results are shown in the table 2.3.

Table 2.3: DeepQA results

	Precision	Recall	F_1
DeepQA	44.8 %	53.7 %	44.1 %
DeepJQA	48.0 %	56.9 %	47.1 %
DeepRQA	50.2 %	53.2 %	47.0 %
DeepJRQA	53.1 %	65.0 %	53.3 %

2.3.4 Results comparison

The Table 2.4 summarizes and compares result from all three articles on WebQuestions dataset.

Table 2.4: Results comparison

	Precision	Recall	F_1
Aqqu	–	65.9 %	49.4 %
STAGG	52.8 %	60.7 %	52.5 %
DeepJRQA	53.1 %	65.0 %	53.3 %

2.4 YodaQA approach

Current YodaQA approach for retrieving answer from the structured knowledge base uses a different approach than the previous articles. The train split of the WebQuestions dataset is used to train a model as well as in the described articles. To train the model, firstly, the gold standard of the paths needs to be generated from the question – answer pairs. Even though, this step requires the entities in the question to be identified, the entity linking process is not applied. Instead, the list of Freebase keys, which are distributed with the WebQuestions dataset is used for each question. The Freebase key is a human readable ID which uniquely identifies the entity.

Using these IDs, all properties for each entity can be explored. If the answer is found after exploring first level property, the node it leads into is not expanded anymore. Otherwise, the exploring continues until two property levels are explored. For each question, all property sequences leading to the answer node are stored as a property path gold standard.

Once the gold standard is generated, it can be used for a model training. In this case, a multi-label linear classifier is used (logistic regression) to predict the relation paths. The model takes LATs and SV generated during question analysis as an input. The outputs are the probabilities of the individual paths. Top N paths (usually 15) are used for retrieving the answers from Freebase. This approach allows prediction of only those paths which were included in the training dataset.

Table 2.5: YodaQA results

	ACC @ 1	AP-recall	MRR
Train	49.8 %	66.9 %	0.558
Test	39.7 %	60.3 %	0.464

The table 2.5 shows results of the YodaQA pipeline using this approach. The evaluation of the YodaQA pipeline uses different metrics than the ones used in the previous articles. The metrics are: accuracy at 1, AP recall and MRR.

Chapter 3

Implementation

This chapter describes the implementation of own approach of generating answers to the questions in natural language from the structured knowledge base. This implementation is a part of the YodaQA pipeline. This chapter is divided into several sections. At the beginning, the dataset needs to be generated because it will be used in all others parts of the chapter. Simple extension based on the current YodaQA approach is then described in order to be comparable with the other efforts. Next two sections describe a the dot product based approach for the relation scoring and a scoring using neural networks respectively.

3.1 Overview

The answering questions using the structured knowledge bases is beside retrieving the answers from the unstructured sources one of the approaches for retrieving the answers in the YodaQA system. The part of the system which produces the answers (from an arbitrary source) depends on the question analysis part, where the entity linking step is done. The Figure 3.1 illustrates relevant components of the pipeline as well as the services called from the pipeline components.

The figure does not contains all YodaQA components. It only aims to those parts of the system, which are essential or developed in this work. The part developed in this thesis is in the middle of the image labelled *Answer producing* including the *Scoring-API*.

The image shows three important segments of YodaQA: Question analysis, Answer producing and answer merging/scoring. They all uses a CAS (common analysis system/structure) for storing multiple annotations. The components illustrated as a cloud shape are those parts of the system which are implemented in Python and communicate with the base Java application over the REST API. This separation is made for two reasons:

1. The required libraries are in Python and it would be impossible or very difficult to simulate the same behaviour in Java code.

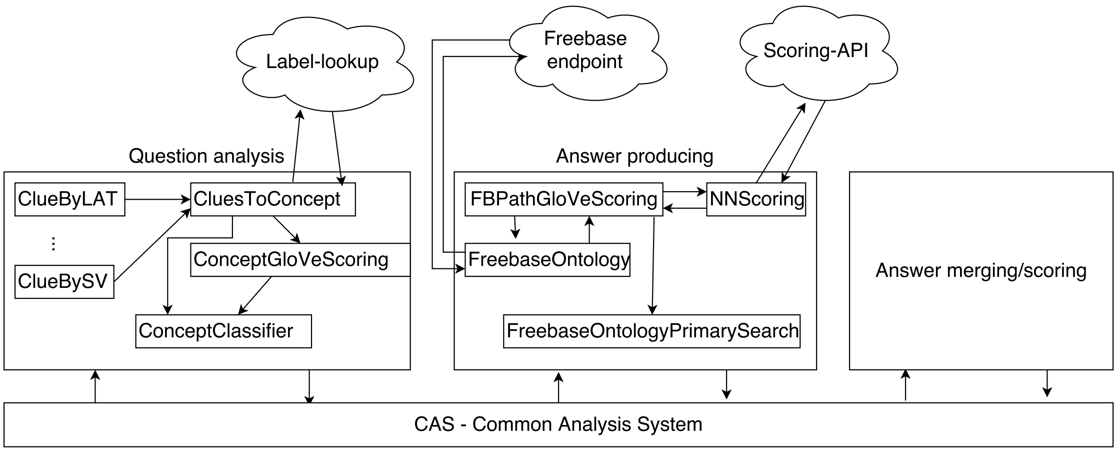


Figure 3.1: Pipeline structure

2. The module requires a significant amount of resources (typically RAM) and that is the reason it is more convenient to move it to another physical machine.

3.1.1 Dataset

All models were trained using the WebQuestions dataset introduced in the *Semantic Parsing on Freebase from Question-Answer Pairs* article[15]. The question candidates used for building the dataset were obtained using Google Suggest API. Random 100 thousand questions out of 1 million candidates were submitted to Amazon Mechanical turk, where users were asked to answer those questions using only entities found in Freebase page of the question topic entity.

The final dataset contains 5,810 questions, 3,778 in the training split and 2,032 in the testing split. Originally provided dataset[3] contains only a natural language questions, a list of the answers and the URL to Freebase page. The modified dataset stored in Brmson Github repository[4] adds some additional information such as the question IDs and introduces different splits. Beside the topic-based splits, more fine-grained splits are included. The original train split is divided into three separated ones:

1. *Devtest* – used for the development e.g. feature selection (189 questions)

2. *Val* – used for the model validation and tuning (755 questions)

3. *Trainmodel* – used for the actual model training (2834 questions)

Since the *devtest* split is not used in this work separately, it is combined with the *val* split into the validation split with size of 944 questions.

[3]http://www-nlp.stanford.edu/software/sempre/

[4]https://github.com/brmson/dataset-factoid-webquestions

The mentioned Github repository contains a branch named “movies”. It is the WebQuestions dataset extended by the moviesE dataset[5] (moviesE already contains a subset of questions from WebQuestions dataset, those are not added again). This extended dataset is also be used in this work.

The repository contains additional data related to the questions. Some of them are part of this work and they are discussed in following sections.

3.1.2 Entity linking

The entity linking used in this work is already implemented in YodaQA question analysis part of the pipeline. Its development was not part of this thesis.

First of all, the entity candidates are generated. These candidates are called clues and they are generated from several question annotations. These annotations (some of them described in the first chapter) are: Lexical Answer Type, Named Entity, Subject, Selection Verb and Token Constituents.

Once the clues are extracted, they need to be linked to the Wikipedia articles. This is done using *CrossWiki Search* and *Fuzzy Search*[6].

- **CrossWiki Search** approach was inspired by [9] and [13] articles. It uses a dictionary[7] which contains a string s, Wikipedia URL u and a probability of the URL given the string on every line: $P(U = u|S = s)$. This dictionary was built using multiple sources such as Wikipedia titles and anchor text from Wikipedia links (both internal and external).

  This file is then loaded into a SQLite database. In order to make the queries fast, an index is created on the label column. This speeds up the query significantly from 14s to 0.02s. YodaQA then can ask for the entity label and ID using REST API.

- **Fuzzy Search** was inspired by [16] article. Its goal is to find the correct entity even if the label is not spelled properly. At the beginning, a sorted list of labels is made. The label is then found in the list using binary search. The Levenshtein distance is computed in every string comparison during the binary search process. If it is lower than a certain threshold, it is considered as a match. Alternative Python implementation PyPy is used to speed the queries up.

Once the clues are mapped to the canonical labels and/or Wikipedia pages IDs, the search process is then repeated on the n-gram clues. This is a special type of clue generated considering all tokens from the sentence. Then 2, 3 and 4-grams of those tokens are made and sent to the label lookup service. If some of these clues cover some of the shorter ones, it is dropped.

Finally, linked clues are called concepts and only 5 with the highest score are used in the following steps (unless different number is specified in the system property).

[5]https://github.com/brmson/dataset-factoid-movies

[6]https://github.com/brmson/label-lookup

[7]http://nlp.stanford.edu/data/crosswikis-data.tar.bz2/dictionary.bz2

3.1.3 Neighborhood exploring

In the Section 2.2, the problem of answering questions from the structured knowledge bases was described as a finding "path" of the relations from the topic entity node to the answer entity node. This subsection describes the process in more detail.

As the gold answers to the questions from the dataset were generated using Freebase, we use Freebase as an answer source as well. The answers from the gold standard could be found in the mentioned knowledge base with two main exceptions: Some of the questions can have some answers containing typos or incomplete information. Some questions can have slightly different answers in more recent Freebase dumps than they had when the dataset was generated.

The task consists of two significant parts. The first one of them is the entity linking and it was described in the previous subsection. The second one is called relation matching. In this step, we try to identify the correct relations or sequence of relations. In a graph theory, the entities are nodes, the relations are edges and the correct sequence of relations is the path from topic entity to the answer entity.

The answer node is the object we want to find. We need to establish a path that is likely to lead to this node. Given the entities provided in WebQuestions dataset, the answer nodes can be reached using a single relation in 55 % questions from the training split and using two relations, we can achieve the answer nodes in 36 % questions. These paths can generate incorrect answers beside the correct ones. This is tested in detail in Experiments chapter. To determine the correct relation sequence, all possible relation sequences are listed and ranked in order to find the correct one.

Neighbourhood exploring is an important part of the process. It is used to generate the gold standard and the candidates for scoring during the actual question answering process. The inputs to this process are all entities retrieved during the entity linking step. Each entity has multiple properties (relations), which connect them to the other entities or the text nodes.

As described in this subsection, 91 % of the questions in the training split of the dataset can be answered using a sequence of the relations with the maximal length of 2. Listing all possible relation sequences will be done in the following way:

- Given the entity e, every relation r is listed if the triplet (e, r, m) exists in the knowledge base KB for any node m which is not CVT.

$$p_e^1 = \{r | \forall r, \exists m : (\text{type}(m) \neq CVT \land \exists(e, r, m) \in KB)\}, \forall e$$

- If the node m in the triplet (e, r_1, m) is CVT, we list all relations r_2 if the triplet (m, r_2, a) exists in the the knowledge base KB.

$$\begin{aligned} p_e^2 = \{(r_1, r_2) | \forall r_1, r_2, \exists m, a : (\text{type}(m) = CVT \land \exists(e, r_1, m) \in KB \\ \land \exists(m, r_2, a) \in KB)\}, \forall e \end{aligned}$$

- The previous case can be extended adding a constraint relation if the question contains more that one entity. We call this constraint a *witness* relation and it is used to reduce the set of the answers generated using the relation path without this constraint.

$$p_e^3 = \{(r_1, r_2, r_3) | \forall r_1, r_2, r_3, \exists m, a : (\text{type}(m) = CVT \land \exists (e_1, r_1, m) \in KB \\ \land \exists (m, r_2, a) \in KB \land \exists (m, r_3, e_2) \in KB)\}, \forall e_1, e_2 : e_1 \neq e_2$$

The CVT node is used to describe the relationship between two entities if it is not possible to describe it using a single relation. It is because the relationship needs to be described using multiple information. The example could be a marriage of two people. The CVT node represents the marriage and have associated relations to wife, husband, date of the wedding and possibly to other nodes.

The Figure 3.2 illustrates three described types of the paths. Note that the third type is not a path in the graph theory language but it forms a tree. We call this a "branched" path.

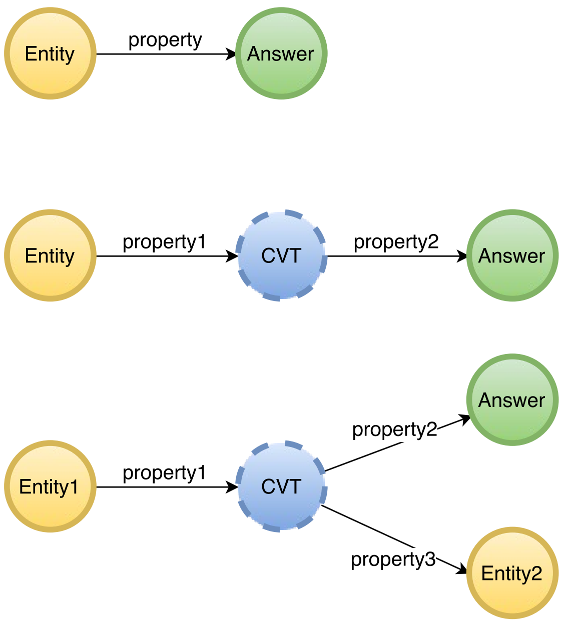


Figure 3.2: Three basic query structures

Following list shows examples of each type of the relation path using Freebase relations. These questions were selected from the training split of the WebQuestions dataset:

1. *what currency do mexico use?* – the answer to the question can be found using a single relation: location.country.currency_used. Corresponding SPARQL query is (without prefix declaration):

```
SELECT ?answer WHERE {
  %entity_id% ns:location.country.currency_used ?answer .
}
```

2. *what is the name of justin bieber brother?* – because the sibling relations contains multiple information, the entities are connected using the CVT node and two relations are needed: people.person.sibling_s and people.sibling_relationship.sibling. SPARQL query:

   ```
   SELECT ?answer WHERE {
     %entity_id% ns:people.person.sibling_s ?m .
     ?m ns:people.sibling_relationship.sibling ?answer .
   }
   ```

3. *who plays meg in family guy?* – there is an additional witness relation as we want the actor of a specific character instead of all the actors in the TV show: tv.tv_program.regular_cast, tv.regular_tv_appearance.actor and tv.regular_tv_ appearance.character. SPARQL query:

   ```
   SELECT ?answer WHERE {
     %entity_id% ns:tv.tv_program.regular_cast ?m .
     ?m ns:tv.regular_tv_appearance.actor ?answer .
     ?m ns:tv.regular_tv_appearance.character %witness_id% .
   }
   ```

3.2 Generating gold standard

Because we want to rank all possible relation paths for a given question (more accurately for all entities obtained for a given question), we need to generate a dataset of these paths. This process uses the dataset described in the Subsection 3.1.1. The questions with the correct answers are stored in `main` directory and the topic entities are in `d-freebase` directory. All files are in JSON format and provide a unique ID of the questions. All scripts which generate auxiliary files and the actual dataset in requested format are located in `scripts` directory.

The two top-level scripts `dump-refresh.sh` and `propsel-dataset-refresh.sh` regenerates all the important files and build the final dataset. These are only wrappers of Python scripts doing the actual work.

As a first step, the entities contained in each question need to be generated. This is done using YodaQA *questionDump* task. It runs the question analysis part of the pipeline and saves selected annotations into a file. The file then contains SV, LATs, Clues and Concepts. Information about concepts (entities) are important for this task. The concepts contains labels, Wikipedia page IDs, score, label probability, description, starting and ending position in the sentence and additional features.

Once the dump containing entities is created, we can start building the gold standard of Freebase paths. It can be done in two ways:

- Using Freebase API, we can request all information about a single entity if we know its ID. As a response, we obtain a JSON containing all information about the entity. The JSON file contains object with a key "property" which consists of all properties associated with the entity. The individual properties are objects where the keys are their names and the values are text, ID (if its available) and type of the node where it leads to. If type of the node is "compound", it also contains object "property" and it can be traversed recursively.

The file is iterated over all properties and if the property has a string contained in "text" field equal to some answer string, the property is saved as a length 1 path into the gold standard file. If the property leads to a compound node, it is traversed recursively. This process finds paths of length 2. During exploration of the second level properties, we also check whether it leads to some other entity from the question. If it does, we save it as a path suffix and if a property leading to the answer is found, this suffix is appended. This check is done comparing the entity IDs if they are available or comparing the entity labels.

The extracted paths are saved in `d-freebase-rp` and `d-freebase-brp` directories. It depends on whether we want the witness relations to be generated (brp) or not (rp). Each file is named according to the split (trainmodel, test, val, devtest) and it contains the question ID and the list of the paths leading to the answer. Each path also has a number associated which tells how many times the path was found in JSON file of the entity.

Unfortunately, Freebase API was shut down at the beginning of May 2016 (unavailability was discovered May 4th 2016). It provided a very fast way to generate the gold standard of the paths as well as the dump of all relations corresponding with the given entity. Fortunately, we cached all JSON files[8] while these scripts were used. However, if the entity linking finds new entities, these will not be able to be explored using this method.

- Running YodaQA *ExploringPathDump* task executes the question analysis and the answer producing part of the pipeline. The property paths are generated in the same way as it is described in the Section 3.4. This only dumps all relation paths for each entity in the question and then the correct ones need to be identified. It also introduces a lot of issues related to the speed of our Freebase endpoint. A lot of relations are blacklisted as they make the query to take a significant amount of time. These relations are relevant to some entities but generate a lot of noise for the others.

3.2.1 Dataset for dot product based scoring

The dot product based scoring estimates a score for each property contained in the paths separately. The scripts doing this job are located in YodaQA repository in the folder `data/ml/fbpath-emb`.

As a first step, first and second level relations need to be generated separately. Script `generate_relations.py` serves for this purpose. First parameter specifies whether we

[8]http://pasky.or.cz/dev/brmson/fbconcepts-2016-05-04.tar.gz

want to generate 1st or 2nd level relations. The levels need to be generated in increasing order as they use the information about lower levels. This script runs fast for first level but it takes a significant amount of time to generate second level relations because there are a lot of nodes in the knowledge base that need to be explored. Alternatively, this could be generated from the files described in the very next subsection. The full paths only need to be split into the individual properties.

The samples for each question are stored in the separated files. They are generated using `fbpath_emb.py` script. This uses the gold standard paths located in *dataset-factoid-webquestions* repository. The question is represented by LATs and Selection Verb and relations are represented by their label.

3.2.2 Dataset for neural network based scoring

The dataset for the training the neural networks needs to be generated in a comma-separated format. It needs to contain the a positive and the negative question – relation path pairs. Each line is in a format `question,label,relpath`. Generating of the positive examples was described earlier in this section. Negatives ones are generated in almost same way (exactly same way using YodaQA pipeline). The script which generates the path leading to the answer nodes now does not need to check whether the text in the node is the same as in some answer. However, it stores all the paths to a file instead.

Generated property paths contain only property names but we want to use property labels to train the model. This is done by simple SPARQL query shown in the Figure 3.3.

```
SELECT DISTINCT ?proplabel WHERE {
  %property_name% ns:type.object.name ?proplabel .
  FILTER( LANGMATCHES(LANG(?proplabel), "en") )
}
```

Figure 3.3: Querying property label

Now we have all the data required to construct the CSV file. All words from both the question text and the property labels are saved in lower case (WebQuestions are in lower case anyway). The correct examples have label 1 and the incorrect ones have label 0. The property labels for individual properties from paths longer than 1 are separated using "#" symbol.

Two variants of the dataset can be generated:

1. The question text is saved "as is" with no additional modifications.

2. All entities which were used for generating corresponding property path (both topic and witness entity) are replaced by the "ENT_TOK" token. To determine which part of the sentence needs to be replaced, we use information about starting and ending position of each entity stored in the question dump file.

If we replace the entities in the way it was described, we can get contradicting samples. It can happen if two question differ only in entity word. We simply remove negative samples of these questions.

3.3 Extending original YodaQA approach

The original approach, as described in the previous chapter, uses simple logistic regression multi-label classifier to predict the relation path. The implementation also brings these two disadvantages:

1. Entity linking is not employed in this process
2. Only direct path from the topic entity to the answer entity is considered (no witnesses)

Only one entity represented by the Freebase page for every question in WebQuestions dataset is used. Since the answers to the respective question were obtained using this entity, it is guaranteed for the path leading from this entity to the answer to exist (with some exceptions such as the question with incorrect answer annotations). However, these entities does not have to be found during the entity linking process and the query based on predicted correct path does not necessarily have to find the correct answer using different (incorrect) entity. This extension uses entities from YodaQA entity linking plus the the correct one from the WebQuestions dataset.

A single property or two properties sharing one CVT node are the only two options considered as a direct path from the topic entity to the answer entity. For some kinds of these questions, multiple answers can be generated in addition to the correct one(s). Typical example of such question is: *Who plays Marge in the Simpsons?*

The gold standard of the Freebase paths (including witness relations) is used to train the multi-label logistic regression classifier. The question is represented by LATs, SV plus Subject. They are all string features. The vocabulary is created from these strings. Each string is then associated with the index in the vocabulary. The vector representing the question contains 1 on index which is associated to the string features extracted from the question and 0 otherwise. Each relation path leading to the answer node is considered as a label (class).

The model is trained using Python module Scikit-learn[17]. The learning strategy is one versus rest, which means that one classifier is learned for each class.

3.4 Generating candidates

The property path candidates need to be generated and ranked using selected model. This step is part of the answer producing step in the YodaQA pipeline. The exact same approach which uses Freebase API is employed in YodaQA as it is described in the Section 3.2. Since the Freebase API is shut down, another method is triggered when some of the concepts from the question is not stored in the local cache of Freebase entities.

This approach introduces some modifications such as blacklisting some properties which helps the query to be processed faster. Unfortunately, some of them can cause losing the possibility of getting the correct answer. However, the filtration of the properties needs to be applied, otherwise some queries would consume significant amount of time (in the matter of hours).

The process starts after the question analysis part is finished. It takes all linked concepts and tries to generate the relation path candidates using the cached data from Freebase API. Every concept which is not found in the cache is stored for the additional exploration. For each of those concepts, the query for retrieving all relations which have the concept as a subject is constructed. We call these the concept relations. This query contains filter which drops all blacklisted properties (described later). After that, another query retrieves the labels for each property separately. These two queries could be merged into one but the experiments showed that separated queries run faster. If the selected method of paths scoring computes score for each property separately, the question and the property representation are sent to the model and the score is estimated.

For all concept relations, we retrieve ID of an object they lead into. If the object has an empty label we know that it is a CVT and we continue in the exploring. Otherwise, we do not use this concept relation for the exploration of the longer paths. However, the ID of CVT node itself is not used for the query construction used to obtain the second relation. The query is based on the concept ID and the concept relations (Figure 3.4) as it can lead to a lot of nodes with the same properties. Doing it this way, we need to process smaller results of unique properties instead of large number of the CVT nodes. Also, the score is computed if we use separate scoring.

```
SELECT DISTINCT ?prop WHERE {
  %concept_id% %concept_property% ?cvt .
  ?cvt ?prop ?val .
}
```

Figure 3.4: Querying second level property

For each question containing two and more entities, we take each pair of them. We build this pairs sensitive to the order of the entities – (e_1, e_2) and (e_2, e_1), where $e_1 \neq e_2$, are two different pairs. The first entity is called concept entity and the second entity is called witness entity. Then we try to find out whether they are connected strictly by two relations with one CVT node between them. If we succeed, we compute a score for the relation between CVT and the witness node. For all paths of length 2, we generated in the previous steps, we check whether there is an intersection between their concept relation and witness candidate concept relation. In addition we make sure that second relation from path and witness relation from witness candidate is not the same. If we found such pair, we add witness relations to the corresponding path.

Now we have all relation paths candidates generated. Two types of the final scoring are possible:

1. If the scores were computed separately for each relation, we need to estimate the final score for full path. Two approaches are introduced:

    a) Average of all scores. This approach often prefers shorter paths over the longer ones. We want to prefer paths with the witness relation over the paths without it if possible. If the witness relation have low score, the average is decreased and shorter path wins.

    b) Comparing score for each relation separately. If scores of the first relations are the same we continue to the second relations (otherwise we return path with higher one) etc. If the paths have different lengths and the scores for relations are the same, we return the longer path.

2. The score is computed for the full path. The model takes representation of question (with or without entity replacement) and the relations from the path separated using “#” symbol and it returns single score for the path.

As it was mentioned before, some properties are blacklisted. This blacklist is used in every SPARQL query which is likely to produce a large list of results. The first set of blacklisted properties was selected using YodaQA run on WebQuestions dataset and the properties which appeared more than 20 times in the results of at least 2 concepts were blacklisted. This is relatively strict criterion so further experiments were implemented. Blacklisted properties were sorted according to the count of occurrences in the gold standard. The top ones were removed from the blacklist and the query time was examined. The example of a property which can make the query to take enormous long time is `location.location.people_born_here`. For example, given the concept of the USA, it lists all people born in the USA which can take hours using our Freebase endpoint. Simple limit on number of results does the effect of reducing time but a lot of information can be lost.

3.5 Dot product based selection

The dot product based property selection uses logic implemented in *Sentence-selection* repository[9]. It is inspired by *Deep Learning for Answer Sentence Selection* article[18]. The dataset for this task is described in the Subsection 3.2.1. The dataset contains question representation (LATs, SV, Subject) and relation representation (label). Both representations can consist of one or more words. We need to transform these representations to the fixed length vectors. For this purpose, we use pre-trained 50-dimensional GloVe[19] vectors. The GloVe vectors are trained on the large corpus of text with goal to obtain a vector representation of words where distance between the vectors representing semantically similar word should be small.

[9]https://github.com/brmson/Sentence-selection

We compute the vector representation for each word from the question representation. We want to obtain a single vector rather than a sequence of vectors. Therefore, we compute average of all vectors. The same method is applied on the relation representation. Now, we have vector $\mathbf{q}$ representing the question and $\mathbf{p}$ representing the relation (property). Given the dataset of positive and negative pairs of the vectors, we want to train a model which returns probability of a label y being 1 given vectors $\mathbf{q}$ and $\mathbf{p}$.

$$P(y = 1|\mathbf{q}, \mathbf{p}) = \sigma(\mathbf{q}^T \mathbf{M}\mathbf{p} + b)$$

The model is represented by $M \in \mathbb{R}^{50\times 50}$ and $b \in \mathbb{R}$ parameters. It is learned using a gradient descent. We learn three models $(\mathbf{M}_1, \mathbf{b}_1)$, $(\mathbf{M}_2, \mathbf{b}_2)$ and $(\mathbf{M}_3, \mathbf{b}_3)$, one for each relation from the property path $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$ respectively. All coefficients are saved and loaded during YodaQA pipeline in order to obtain a score for the individual relations.

This approach brings some issues such as losing information about the word order and need for a method to create the final score from the separate ones.

3.6 Neural network based selection

The neural network based selection tries to solve the issues described at the end of the previous section. Several types of the neural networks[20] are introduced to be used as a model:

- **CNN** – convolutional neural network
- **RNN** – recurrent neural network
- **AVG** – average of question and path representation
- **ATTN1511** – attention based network[21]

All models, datasets and scripts for training and evaluating are stored in *dataset-sts* repository[10]. Implementation uses Python framework *Keras*[22].

All described models share the same structure for input and for score estimation from the learned representations. The dataset used for learning and evaluation is described in the Subsection 3.2.2. As a first step of all tasks (training, evaluating, scoring), a vocabulary is made from the training split of the dataset. The vocabulary consists of the tokens created from both the question and the property path representation. The input into the model is a vector of indices of the words from the vocabulary. There are separate vectors for the question and the property path, both with fixed length of 60 elements.

Words represented as an index into the vocabulary are then transformed using 300-dimensional GloVe[19] vectors. This results into the matrix with 300 rows and 60 columns. This representation servers as an input into a model specific layers.

[10]https://github.com/brmson/dataset-sts

The output from the model specific layers is then fed into MLP scorer (layers Projection and above in the Figure 3.5 and 3.6). The first layer is a projection which is a fully connected layer. It typically shares weights over the question and the property path if not specified otherwise. Outputs of this layer for the question and the property path are summed and multiplied element-wisely. The outputs from the sum and multiplications are then concatenated and fed into the fully a connected layer which reduces the dimensionality. This is followed by last the layer which has a single value as an output.

3.6.1 Convolutional neural network

The convolutional neural network uses multiple channels of convolutions with different lengths (1 to 5-token channels). Each channel consists of $\frac{N}{2}$ convolutions (N = dimensionality of word vector). The channel of length 1 is illustrated by the red color in the Figure 3.5, the yellow color is used for length 2 and the green color for length 3. The stride of each channel is 1, meaning that the channel shifts by one word. The convolutional layer is followed by a max pooling selecting maximum per each channel length and concatenating results. This is then fed into the projection layer.

The convolutional and max-pooling layers can be siamese or non-siamese. A siamese layer means that the weights are shared for the question and the relations paths representation. The siamese network reduce the number of model parameters. However, the experiments showed that non-siamese performs better.

3.6.2 Recurrent neural network

The recurrent neural network makes use of a memory unit in order to learn information hidden in word order. A GRU[23] unit is used as a the memory unit in our model. There are two recurrent layers, one for the forward direction of words and one for the backwards direction. Each one of the layers consists of $2N$ GRU units (illustrated by red color in the Figure 3.6). The outputs of the final units of each direction are then summed up resulting in a vector representation of length $2N$. This output serves as an input into to the projection layer.

As well as in the CNN model, recurrent layer can be siamese or non-siamese. Non-siamese one also performs better. However, the performance boost is not as significant as in the CNN model.

3.6.3 Other models

Other models which were tested are ATTN1511 and AVG. The ATTN1511 model is described in [21] and [20]. The main idea of the model is to prefer some parts of the input sentence when the building sentence representation. This model achieves similar results as non-siamese RNN.

The AVG model uses simple average of the vectors from the input sequence. The vectors are scored using described MLP scorer. This model has poor performance which indicates that more complex network structures are useful for this task. The objectives

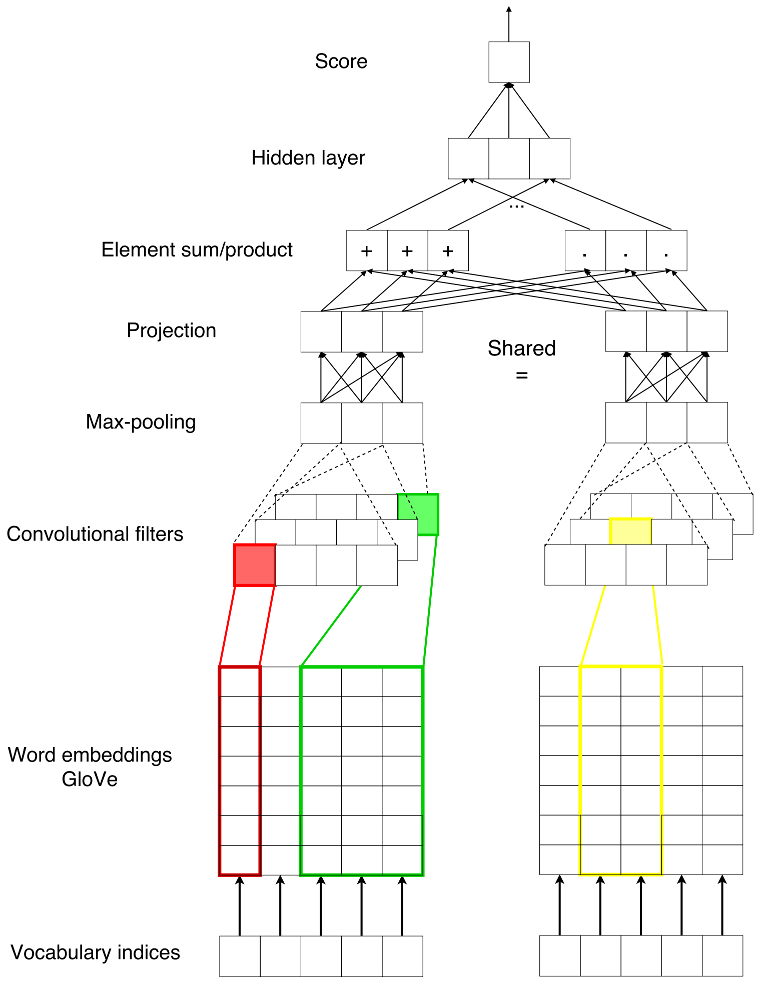


Figure 3.5: Non-siamese convolutional neural network

tested in the training of all models was bipartite version of Ranknet[24] and binary cross-entropy.

3.7 Final answer producing

3.7.1 Scoring API

Once the model is trained, all learned weights are saved into a file. A scoring API is a Python script which takes as an input the model name, training split of dataset used to build vocabulary, file containing weights and model parameters (siamese or not,...). After it starts, it loads the GloVe vectors, builds the vocabulary and the model, loads the weights and starts a REST endpoint which is used in YodaQA pipeline to get the score for the question – relation path pairs. It is not directly implemented in YodaQA

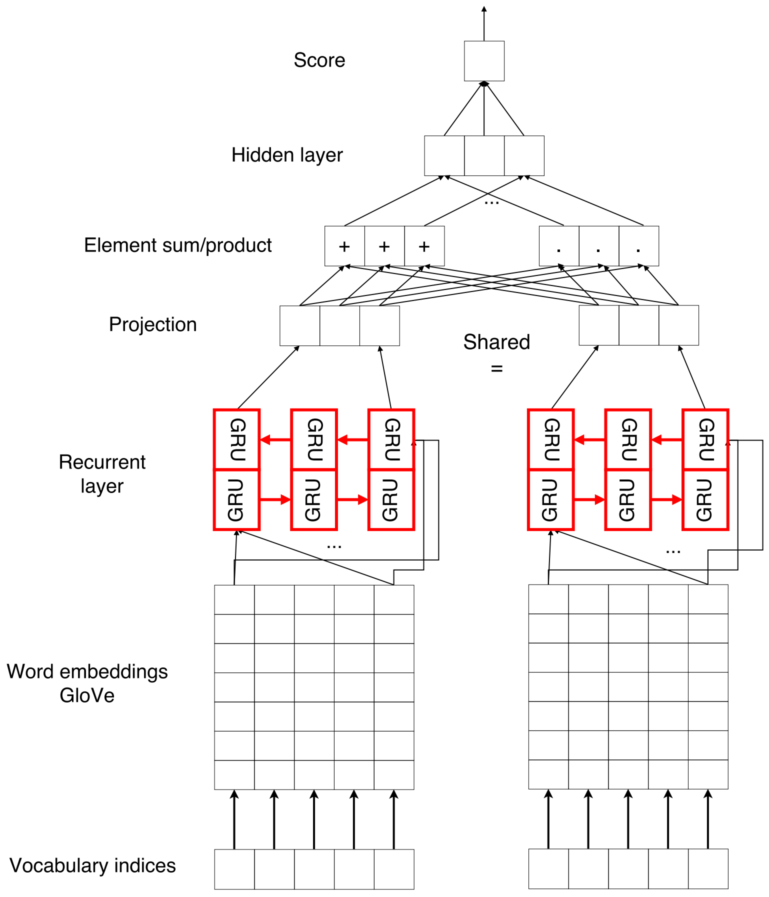


Figure 3.6: Non-siamese recurrent neural network

because it is easier to reuse Keras functionality and required resources can be allocated on a different machine.

3.7.2 Answer scoring and merging

Once the relation paths are scored and sorted, top N of them is selected (the value 15 was experimentally selected). All selected paths are used to produce the answers. Each answer has a set of the features (some of them described in the Subsection 1.3.3). If some answer is present in the list multiple times, the occurrences are merged and the number is used as a feature as well. Finally, answers are scored using Decision Forest and sorted. This is sufficient if we need just accuracy at 1, AP-recall or MRR metrics. If we are required to measure precision and recall, we have to employ a technique which returns only such answers marked as correct.

Chapter 4

Experiments

This chapter is divided into the several sections. At the beginning, we are experimenting with selecting subsets of the gold standard paths. After that, the logistic regression based path selections are compared. Next two sections describe the performance of the question answering system using the dot product based scoring and the neural network base scoring.

4.1 Dataset experiments

The dataset of the gold standard Freebase paths was described in the Subsection 3.1.1. It contains every path which leads to the answer node regardless of the number of the incorrect answers it generates. The datasets described in this sections are WebQuestions (WQ) and WebQuestions + moviesE (WQM).

4.1.1 Empty gold standard of the paths

Some of the questions do not have any gold standard path generated at all. The numbers of the empty paths differ whether we used the entity linking or only the entities provided in the dataset. This difference only applies to the WQ dataset because the movies datasets do not provide the gold standard entities. The Table 4.1 shows how many questions from the dataset do not have any gold standard path according to the used entity source (GSE - gold standard entities, LE - linked entities). The experiments are done on the training split of the datasets.

The result of the empty gold standard path list is that we cannot learn the correct relation path for the particular question sample. Note that the high number of the empty paths in WQM / GSE line is because the moviesE questions have no list of the entities provided. Additionally, the gold standard entity was not found in 620 questions out of 3778 (16.4 %) using the WQM dataset. We use WQM / GSE + LE for further model training. We include the gold standard entities not found by YodaQA entity linker to increase path coverage.

Table 4.1: Empty gold standard paths

Dataset / used entity	Question count	No path	Percent
WQM / GSE + LE	4573	492	10.7 %
WQM / GSE	4573	1140	24.9 %
WQM / LE	4573	919	20.0 %
WQ / GSE + LE	3778	336	8.8 %
WQ / GSE	3778	345	9.1 %
WQ / LE	3778	763	20.1 %

Error analysis

As shown in the Table 4.1, the path from the topic entity to the answer entity was not found for all questions. Here there are 10 randomly selected questions with no correct path generated considering that the failure was not caused by the incorrect entities:

- *what was malcolm x trying to accomplish?* – The answers "African Americans' rights" and "Black Liberation" were not found in the entity neighbourhood. The first answer only occurs in the topic description. The closest answer found is "Human rights activist".
- *what money is used in england?* – The answer in the Freebase is "Pound sterling" instead of the requested "UK £".
- *what year michael jordan came in the nba?* – The answer "1984" can be found in the Freebase, but the gold standard answer is "1984 NBA Draft".
- *when did mary shelley write frankenstein what were the circumstances?* – The answer in the gold standard is a whole description which is not considered as an answer source.
- *what is 2pm est in philippines?* – The requested answer is "UTC+8" but "Philippine Time Zone" was found which is an equivalent answer.
- *what round did manny pacquiao win in?* – The gold standard contains a wrong answer "celebritynetworth.com".
- *what year did william mckinley became president?* – Only a year is requested but the Freebase contains a full date "1897-03-04".
- *what is the timezone in england called?* – The gold standard answer is "Greenwich Mean Time" and the Freebase contains "Greenwich Mean Time Zone".
- *what was malcolm x trying to accomplish?* – The gold standard answer is "AIDS" and the Freebase contains "HIV/AIDS".
- *what disease did helen keller?* – No relation regarding diseases is contained in our Freebase dump

4.1.2 Gold standard reduction

Since all paths leading to the answer node are used as the positive examples, some of them can generate a lot of incorrect answers. For example, the question about a director of a film has the “directed_by” as a correct relation. However, the relation “award_nominee” could also lead from the film node to the director node. Additionally, the relation can generate more people who are not the directors of the requested film.

The paths are paired with the entities in order to obtain an answer. We compute precision p, recall r and F_1 score for each path – entity pair. Informally, precision says how many of the answers marked as a correct were in the gold standard and recall says how many of the gold standard answers we discovered. The formulas for these metrics are following:

$$p = \frac{TP}{TP + FP} \qquad r = \frac{TP}{TP + FN} \qquad F_1 = \frac{2 \cdot p \cdot r}{(p + r)}$$

A variable TP means true positive (number of correct answers marked as correct), FP means false positive (number of incorrect answers marked as correct) and FN means false negative (number of correct answers marked as incorrect).

We want to sort all paths for each question separately according to the F_1 score. Since the path can generate multiple sets of answers given the different entities, we compute average and maximum over all answer sets for each path. Then we select only those paths which have the highest average and maximum F_1. If no path has highest both of the values, we take the paths with the highest average of F_1. Although, we compute the average and maximum F_1 for each path individually, we report mean values $\bar{F_1}$ and ${F_1}^*$ over all path and questions. If we use the original dataset without the reduction, the ${F_1}^*$ value is 0.697 and the $\bar{F_1}$ value is 0.562. If we filter the paths as described earlier in this paragraph, the values are 0.744 and 0.647 respectively.

The intuitive meaning of the maximum F_1 value is the upper bound of the F_1 value achieved at the end of the question answering process. This also requires that the correct entity was selected. The meaning of the average F_1 is the same with the difference that we use all linked entities to produce the answers.

4.1.3 Property blacklist

A lot of properties were blacklisted for the performance issues as described in the Section 3.2. The WQM gold standard of the Freebase paths contains 11,870 properties and 813 of them are blacklisted. It affects 704 out of 4573 questions (15.3 %). The Reduced WQM gold standard of the paths contains 7,524 properties and 566 of them are blacklisted which affects 557 out of 4573 questions (12.3 %). If the question if “affected” by the blacklisted property path, it means that one or more its property paths contains the blacklisted property. It does not necessarily means that the correct answer cannot be found. However, if the question contains only one correct property path, then the path cannot be used and no answer is found.

Table 4.2: Blacklisted properties

Blacklisted properties	# in GS paths	# in reduced GS
people.person.profession	136	119
location.country.languages_spoken	122	58
location.location.contains	102	54
travel.travel_destination.tourist_attractions	80	77
book.book_subject.works	48	34
book.author.works_written	36	28
location.location.time_zones	33	33
symbols.name_source.namesakes	26	15
book.author.book_editions_published	21	9
location.citytown.postal_codes	19	19

The Table 4.2 shows the blacklisted properties in YodaQA sorted according to the occurrences in the gold standard paths. Only top 10 paths are shown. Full table can be found in the Appendix in the Table B.1.

4.2 Logistic regression

This section provides the results of the YodaQA system using the logistic regression classifier described in the Section 3.3. Two types of the classifier are tested. The original one that has no witness relations and the improved one that uses those relations. The system evaluation uses three datasets: WebQuestions, moviesD and moviesF[11]. The moviesF dataset is a modification of the moviesD dataset including some bug fixes and synthetically generated questions. These two movies datasets were selected as they have a different structure. The moviesD introduces more diversity because it does not contain the synthetically generated questions as the moviesF does.

Table 4.3: Original approach

Dataset	Phase	ACC @ 1	AP-recall	MRR
moviesD	Test	38.1 %	61.9 %	0.446
	Train	54.8 %	60.9 %	0.571
moviesF	Test	34.9 %	52.0 %	0.398
	Train	41.7 %	50.1 %	0.447
WebQuestions	Test	39.7 %	60.3 %	0.464
	Train	49.8 %	66.9 %	0.558

The Table 4.3 shows the results achieved on the mentioned datasets by the original system. Firstly, the system finds the answers of the questions from the training dataset.

[11]https://github.com/brmson/dataset-factoid-movies

After that, the decision forest used for the answer scoring is re-trained and the answers are re-evaluated. The very same classifier is applied on the testing questions. Three metrics are provided:

1. Accuracy at one (ACC @ 1) – a percentage of the questions that have the top answer equal to some of the correct answers
2. AP-recall – a percentage of the questions that have the correct answer in the top 20 (this is because we report results of the last phase of the YodaQA evaluation which uses only top 20 answers)
3. Mean reciprocal rank (MRR) – an average of the reciprocal ranks of all answers for each question

The original approach uses neither the witness relations nor the entities from the entity linking for training the classifier. As the moviesD and moviesF datasets do not provide the topics entities, the relation classifier is trained using only the questions from WebQuestions dataset. This is the reason why the results of the movies dataset in the Table 4.3 are not significantly better as in the following experiments.

Table 4.4: Original approach + entity linking

Dataset	Phase	ACC @ 1	AP-recall	MRR
moviesD	Test	47.3 %	74.2 %	0.555
	Train	71.5 %	77.6 %	0.738
moviesF	Test	56.3 %	78.6 %	0.632
	Train	71.6 %	81.9 %	0.754
WebQuestions	Test	40.5 %	61.2 %	0.471
	Train	51.0 %	68.2 %	0.571

The Table 4.4 shows the results after the entity linking was employed. Given the linked entities, the gold standard of the relation paths can be found even for the movies questions. The performance improvement on the movies dataset is significant. It is because of the presence of the synthetically generated questions in the training dataset while the logistic regression classifier was trained. The Table 4.5 shows performance on the moviesD dataset with the entity linking employed but with the movies questions excluded from the training process. It is clear the entity linking alone cause only a slight improvement.

The Table 4.6 shows the results using both the entity linking and the witness relations. The witness relations introduces an additional improvement because the relations reduces the number of the false positive answers.

Table 4.5: moviesD with entity linking, movies question were excluded from the training of the classifier

	ACC @ 1	AP-recall	MRR
Test	38.5 %	66.5 %	0.462
Train	59.3 %	67.1 %	0.626

Table 4.6: Extended approach

Dataset	Phase	ACC @ 1	AP-recall	MRR
moviesD	Test	51.2 %	73.5 %	0.582
	Train	71.3 %	76.9 %	0.738
moviesF	Test	62.5 %	78.4 %	0.679
	Train	75.8 %	83.5 %	0.790
WebQuestions	Test	42.4 %	61.7 %	0.485
	Train	50.4 %	67.5 %	0.565

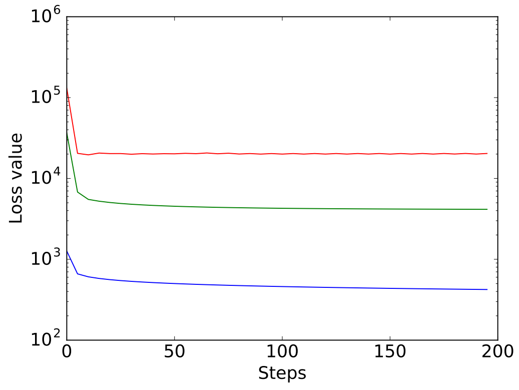
4.3 Dot product based scoring

This section describes the results of the dot product based scoring described in the Section 3.5. Three models, one for each relation type of the property path, are trained. The Figure 4.1 shows the loss value and the MRR development during the learning process. The red line represents learning process of the first (concept) relation, the green line represents the second (expanded) relation and the blue line represents the third (witness) relation.

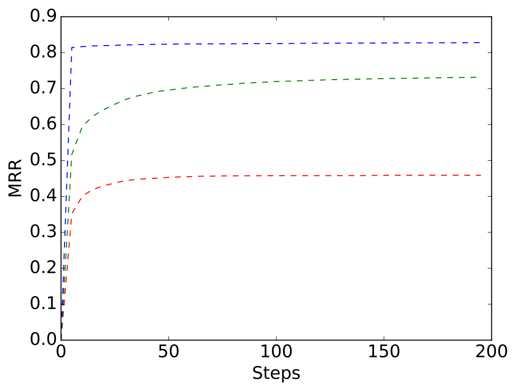
The model is trained using 6-fold cross-validation. The curves shown in the graph represent the mean values over all folds. The final mean MRR is 0.446 for the first relation, 0.712 for the second relation and 0.827 for the third relation. Every question has a concept relation and it has a lot of those relations in its neighbourhood. If the question has the expanded or the witness relations in its neighbourhood, the number of those relation is small and it is easier to recognize the correct one among them. The Subfigure 4.1a uses a logarithmic scale on the y axis.

The Table 4.7 shows the results of the end-to-end process using the dot product based property path scoring. The results are not so impressive and it is caused by several aspects. The question representation loses a lot of information contained in the original question. Although, the same representation is used in the previous approach, this model computes an average over all vectors of the words from the representation which causes an additional information loss.

Another aspect is the way of computing the final score. Two approaches are introduced in this thesis, the average of individual scores and the separate sorting. The table shows the results using the average of the scores which performs slightly better than the other approach.



(a) Loss value



(b) MRR

Figure 4.1: Learning curves for the dot product based model

Table 4.7: Dot product based scoring

Dataset	Phase	ACC @ 1	AP-recall	MRR
moviesD	Test	50.8 %	77.3 %	0.580
	Train	66.8 %	75.6 %	0.702
moviesF	Test	52.9 %	74.5 %	0.593
	Train	65.0 %	76.2 %	0.690
WebQuestions	Test	28.8 %	54.7 %	0.363
	Train	36.3 %	56.1 %	0.426

4.4 Neural network based scoring

This section describes the results of the neural network based scoring described in the Section 3.6. Following models are tested: RNN (siamese and non-siamese), CNN (siamese and non-siamese), ATTN1511 and AVG. Those model are also described in the [20] paper. The Table 4.8 shows the models comparison on WebQuestions + moviesE dataset with the entity replacement (WQME) and on the reduced dataset (WQMER). Each entity mentioned in the question is replaced using the token “ENT_TOK”. All models were trained four times and the mean value is shown in the table.

We can see that non-siamese architecture of the models have slightly better performance than the siamese variant. We can also see that the results of the models on the WQMER dataset are worse than on the WQME. This is due to a ratio of the correct and incorrect samples in the datasets. The WQMER dataset has approximately a half number of the correct samples. We chose the WQME dataset for further testing since it has a chance to assign a higher score to not completely correct samples. Those partially correct samples are marked as incorrect in the WQMER dataset. The Table 4.9 shows the results of the end-to-end system using the scoring based on the non-siamese

Table 4.8: Model comparison using the WQME and WQMER datasets

	WQME		WQMER	
Model	train MRR	val MRR	train MRR	val MRR
CNN-S	*0.893261*	*0.708962*	*0.538412*	*0.505810*
CNN	0.802253	0.697644	0.543348	0.508036
RNN-S	0.824593	0.716676	0.513264	0.491681
RNN	0.814869	0.708421	0.566334	0.532597
ATTN1511	**0.900919**	**0.739048**	**0.738079**	**0.620818**
AVG	0.513481	0.506383	0.486454	0.488889

CNN model (CNN-S). However, the CNN-S is not the best model according to the Table 4.8, the experiments during the development showed that the end-to-end performance is slightly better using this model.

Table 4.9: Neural network based scoring

Dataset	Phase	ACC @ 1	AP-recall	MRR
moviesD	Test	55.0 %	80.4 %	0.628
	Train	76.3 %	83.7 %	0.793
moviesF	Test	62.8 %	84.4 %	0.696
	Train	73.2 %	86.6 %	0.786
WebQuestions	Test	43.9 %	71.6 %	0.522
	Train	53.8 %	77.2 %	0.620

We also tried combination of the logistic regression and CNN-S models and it achieved comparable results as the CNN-S model alone. The Table 4.10 shows all three approaches for the relation matching task used separately. The results are shown on the test split of each dataset.

Table 4.10: Relation matching comparison

Dataset	Method	ACC @ 1	AP-recall	MRR
moviesD	Log. regression	51.2 %	73.5 %	0.582
	Dot product	50.8 %	77.3 %	0.580
	CNN-S	55.0 %	80.4 %	0.628
moviesF	Log. regression	62.5 %	78.4 %	0.679
	Dot product	52.9 %	74.5 %	0.593
	CNN-S	62.8 %	84.4 %	0.696
WebQuestions	Log. regression	42.4 %	61.7 %	0.485
	Dot product	28.8 %	54.7 %	0.363
	CNN-S	43.9 %	71.6 %	0.522

The YodaQA returns a list of all answers found in the process sorted according to their scores. It aims to show the correct answer on the first place which is then presented to the user for example in the web interface. However, it is not capable to return a list of the answers marked as correct. In order to preserve answer scoring path of the pipeline, we try to simulate this behaviour in two ways. The first way is to select all answers with the score higher than a certain threshold as a set of the correct answers. The Figure 4.2 shows the dependence of the precision, recall and F_1 score on the answer score threshold. The blue dashed line represents the precision, the green dashed line represents the recall and the red line represents the F_1 score. The highest value of the F_1 score is 0.351 given the threshold equal to 0.53.

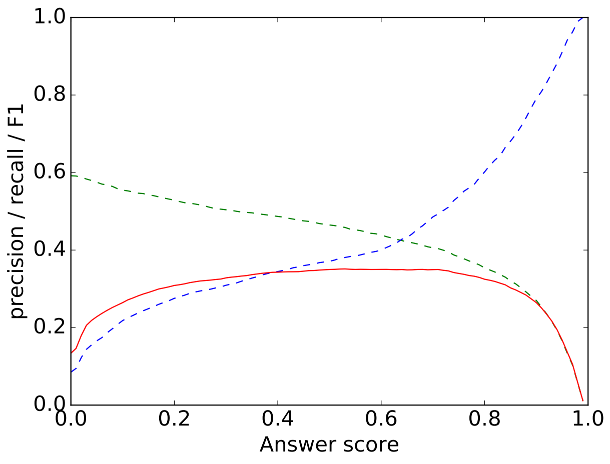


Figure 4.2: F_1 score dependence on the answer score threshold

The other option is based on the property path which generated the answer. We take the answer with the highest score and we add the other answers which were generated using the very same property path. This approach results in F_1 score equal to 0.304. Each of the mentioned techniques is far from the perfection and the YodaQA system needs more sophisticated way to select list of the correct answers. The most straightforward way could be using the top scored path and select all the answers it generates as the correct ones. However, this cuts the process before the answer scoring step which leads to worse results. The Table 4.11 shows the comparison of the F_1 with the solutions described in the Chapter 2.

Table 4.11: F_1 comparison

System	F_1
Aqqu	49.4
Stagg	52.5
DeppQA	53.3
YodaQA	35.1

Conclusion

The goal of this thesis was to improve the question answering performance of the YodaQA system. The selected approach was the usage of the structured knowledge base. The selected knowledge base was the Freebase thanks to its robustness and since the gold standard answers to the questions from the datasets were obtained using this knowledge base. Three most recent articles about the question answering using the structured knowledge base was used as an inspiration.

The selected approach was based on the scoring of the relation paths. The paths were generated using the entities from the entity linking and three scoring approaches were introduced: the logistic regression, dot product based and neural network based scoring. The logistic regression approach was inspired by the previous YodaQA implementation of this task and the simple extension was introduced resulting in performance improvement from 38.1 % to 51.2 % on the moviesD, from 34.9 % to 62.5 % on the moviesF and from 39.7 % to 42.4 % on the WebQuestions dataset.

The dot product based scoring was inspired by the sentence selection task. Unfortunately, the representation of the question and the final score estimation caused this approach to perform worse then the previous one.

The neural network based scoring was inspired by the mentioned articles. The additional models were tested and compared. This approach introduced an additional performance improvement in comparison with the logistic regression approach. The final performance using this scoring method was 55.0 % on the moviesD, 62.8 % on the moviesF and 43.9 % on the WebQuestions dataset.

As mentioned in the previous chapter, the achieved value 0.351 of the F_1 score is very low. The value is low because of the low precision as the YodaQA returns all answers sorted by their score. The future improvements should aim to this issue in order to improve precision as well as to add the ability to return a list of the answers if the question requires it. This could be done for example by improving the entity linking. The Wikipedia description of the entity can be used to measure co-occurrences of the words from the question and from the description as mentioned in [11]. A similar approach can be used for the relation matching. The benefit of the current system is that it needs only the data provided in the dataset for the training which is helpful for a possible domain adaptation.

Another improvement is more about the speed performance of the system. The current Freebase endpoint is very slow and it needs to be replaced with a better endpoint software. Additionally, the Freebase can be replaced with the Wikidata knowledge base since it is not developed and supported any more.

Bibliography

[1] Baudiš, P. YodaQA: A Modular Question Answering System Pipeline. Technical report, Dept. of Cybernetics, Czech Technical University, 2015.

[2] Ferrucci, D.; Lally, A. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Nat. Lang. Eng.*, volume 10, no. 3-4, Sept. 2004: pp. 327–348, ISSN 1351-3249.

[3] Miller, G. A. WordNet: A Lexical Database for English. *Commun. ACM*, volume 38, no. 11, Nov. 1995: pp. 39–41, ISSN 0001-0782, doi:10.1145/219717.219748. Available from: `http://doi.acm.org/10.1145/219717.219748`

[4] Miller, E. An Introduction to the Resource Description Framework. *D-Lib Magazine*, May 1998.

[5] Gandon, F.; Schreiber, G. RDF 1.1 XML Syntax. Technical report, W3C, Feb. 2014, https://www.w3.org/TR/rdf-syntax-grammar/.

[6] Lehmann, J.; Isele, R.; Jakob, M.; et al. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, 2014: pp. 167–195.

[7] Bollacker, K.; Evans, C.; Paritosh, P.; et al. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 2008, ISBN 978-1-60558-102-6, pp. 1247–1250.

[8] Vrandečić, D.; Krötzsch, M. Wikidata: A Free Collaborative Knowledgebase. *Commun. ACM*, volume 57, no. 10, Sept. 2014: pp. 78–85, ISSN 0001-0782.

[9] Bast, H.; Haussmann, E. More Accurate Question Answering on Freebase. In *CIKM*, ACM, 2015, pp. 1431–1440. Available from: `http://ad-publications.informatik.uni-freiburg.de/freebase-qa.pdf`

[10] Yih, W.; Chang, M.-W.; He, X.; et al. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. July 2015. Available from: `http://research.microsoft.com/apps/pubs/default.aspx?id=244749`

[11] Xu, K.; Feng, Y.; Reddy, S.; et al. Enhancing Freebase Question Answering Using Textual Evidence. *CoRR*, volume abs/1603.00957, 2016. Available from: `http://arxiv.org/abs/1603.00957`

[12] Manning, C. D.; Surdeanu, M.; Bauer, J.; et al. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014, pp. 55–60.

[13] Spitkovsky, V. I.; Chang, A. X. A Cross-Lingual Dictionary for English Wikipedia Concepts. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey: European Language Resources Association (ELRA), may 2012, ISBN 978-2-9517408-7-7.

[14] Yang, Y.; Chang, M. S-MART: Novel Tree-based Structured Learning Algorithms Applied to Tweet Entity Linking. In *ACL (1)*, The Association for Computer Linguistics, 2015, pp. 504–513.

[15] Berant, J.; Chou, A.; Frostig, R.; et al. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2013, pp. 1533–1544.

[16] Yao, X. Lean Question Answering over Freebase from Scratch. In *Proceedings of NAACL Demo*, 2015. Available from: `http://cs.jhu.edu/~xuchen/paper/scratch-qa.pdf`

[17] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.

[18] Yu, L.; Hermann, K. M.; Blunsom, P.; et al. Deep Learning for Answer Sentence Selection. *CoRR*, volume abs/1412.1632, 2014. Available from: `http://arxiv.org/abs/1412.1632`

[19] Pennington, J.; Socher, R.; Manning, C. D. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. Available from: `http://www.aclweb.org/anthology/D14-1162`

[20] Baudiš, P.; Šedivý, J. Sentence Pair Scoring: Towards Unified Framework for Text Comprehension. *CoRR*, volume abs/1603.06127, 2016. Available from: `http://arxiv.org/abs/1603.06127`

[21] dos Santos, C. N.; Tan, M.; Xiang, B.; et al. Attentive Pooling Networks. *CoRR*, volume abs/1602.03609, 2016. Available from: `http://arxiv.org/abs/1602.03609`

[22] Chollet, F. Keras. `https://github.com/fchollet/keras`, 2015.

[23] Britz, D. Implementing a GRU/LSTM RNN with Python and Theano. 2015, accessed: 2016-05-18. Available from: `http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/`

[24] Burges, C.; Shaked, T.; Renshaw, E.; et al. Learning to Rank Using Gradient Descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, New York, NY, USA: ACM, 2005, ISBN 1-59593-180-5, pp. 89–96.

Appendix A

Acronyms

QA Question answering

YodaQA Yet anOther Deep Answering pipeline

UIMA Unstructured Information Management Architecture

CAS Common analysis system/structure

LAT Lexical answer type

SV Selection verb

POS Part of Speech

RDF Resource Description Framework

SPARQL SPARQL Protocol and RDF Query Language

CVT Compound Value Type

SVM Support Vector Machine

CNN Convolutional neural network

RNN Recurrent neural network

Appendix B

Property blacklist

Table B.1: Blacklisted properties

Blacklisted properties	# in GS paths
people.person.profession	136
location.country.languages_spoken	122
location.location.contains	102
travel.travel_destination.tourist_attractions	80
book.book_subject.works	48
book.author.works_written	36
location.location.time_zones	33
symbols.name_source.namesakes	26
book.author.book_editions_published	21
location.location.partially_contains	19
location.citytown.postal_codes	19
music.artist.track	18
location.location.events	18
location.country.first_level_divisions	17
location.country.administrative_divisions	16
music.artist.album	14
sports.sports_team_location.teams	12
music.composer.compositions	11
film.film_subject.films	9
visual_art.art_subject.artwork_on_the_subject	6
music.lyricist.lyrics_written	5
periodicals.newspaper_circulation_area.newspapers	5
book.book.editions	4
organization.organization_scope.organizations_with_this_scope	4
food.beer_country_region.beers_from_here	3
influence.influence_node.influenced	3

location.location.people_born_here	3
people.person.quotations	3
music.producer.tracks_produced	3
broadcast.artist.content	2
biology.breed_origin.breeds_originating_here	2
film.film_location.featured_in_films	2
fictional_universe.fictional_setting.fictional_characters_born_here	1
people.profession.people_with_this_profession	1
time.event.includes_event	1
tv.tv_program.episodes	1
media_common.netflix_genre.titles	1
astronomy.celestial_object.locations	1
location.country.second_level_divisions	1
fictional_universe.character_species.characters_of_this_species	1
government.governmental_jurisdiction.agencies	1
award.award.category	0
olympics.olympic_games.events	0
chemistry.chemical_element.isotopes	0
military.military_conflict.military_personnel_involved	0
government.government_office_category.offices	0
music.genre.artists	0
organization.organization_sector.organizations_in_this_sector	0
music.composition.recordings	0
astronomy.orbital_relationship.orbited_by	0
olympics.olympic_games.competitions	0
people.cause_of_death.people	0
tv.tv_genre.programs	0
music.genre.albums	0
music.compositional_form.compositions	0
cvg.cvg_genre.games	0
meteorology.cyclone_affected_area.cyclones	0
film.film_genre.films_in_this_genre	0
business.industry.companies	0
military.military_unit_place_of_origin.military_units	0
award.award_discipline.awards_in_this_discipline	0
media_common.literary_genre.books_in_this_genre	0
wine.wine_region.wines	0
broadcast.genre.content	0
biology.organism_classification.lower_classifications	0
music.genre.subgenre	0
internet.website_category.sites	0

Appendix C

Contents of enclosed CD

```
readme.txt ................................ the file with CD contents description
src .......................................... the directory of source codes
  yodaqa ..........................................................
  dataset-sts .......................................................
  dataset-factoid-webquestions .......................................
  thesis ....................... the directory of LaTeX source codes of the thesis
text ............................................ the thesis text directory
  DP_Pichl_Jan_2016.pdf ......................... the thesis text in PDF format
```