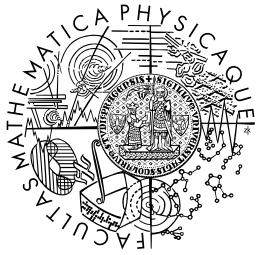


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Ahmad Aghaebrahimian

Hybrid Deep Question Answering

Institute of Formal and Applied Linguistics

Supervisor of the doctoral thesis: RNDr. Martin Holub, Ph.D.
Study programme: Computer Science
Study branch: Mathematical Linguistics (4I3)

Prague 2018

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I would like to express my gratitude to my supervisor Dr. Martin Holub for his valuable feedbacks and continuous help. I also appreciate my ex-supervisor Dr. Filip Jurčíček for his helpful advice and discussions as well as providing me with an excellent opportunity to grow professionally. I am in debt for the enlightening guidance I received through the classes by my senior colleagues Prof. Jan Hajič, Dr. Pavel Pecina, Dr. Zdeněk Žabokrtský, and Dr. Martin Popel too. I would also like to thank my fellow friends Lukáš Žilka, Ondřej Dušek, Natalia Klyueva, and Ondřej Plátek for their many bits of help through my studies at the Institute of Formal and Applied Linguistics.

Last but not least, I would like to express my appreciation to my family for their constant encouragement and support through all these years.

Title: Hybrid Deep Question Answering

Author: Ahmad Aghaebrahimian

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Martin Holub, Ph.D., Institute of Formal and Applied Linguistics

Abstract: As one of the oldest tasks of Natural Language Processing, Question Answering is one of the most exciting and challenging research areas with lots of scientific and commercial applications. Question Answering as a discipline in the conjunction of computer science, statistics, linguistics, and cognitive science is concerned with building systems that automatically retrieve answers to questions posed by humans in a natural language. This doctoral dissertation presents the author's research carried out in this discipline. It highlights his studies and research toward a hybrid Question Answering system consisting of two engines for Question Answering over structured and unstructured data. The structured engine comprises a state-of-the-art Question Answering system based on knowledge graphs. The unstructured engine consists of a state-of-the-art sentence-level Question Answering system and a word-level Question Answering system with results near to human performance. This work introduces a new Question Answering dataset for answering word- and sentence-level questions as well. Starting from a simple Logistic Regression model, the author presents more advanced models including multi-layer Neural Network, Convolution Neural Network and Recurrent Neural Network solutions for different Question Answering systems mentioned above. Although the core of all developed modules in this work are new Deep Neural Network architectures, several linguistic intuitions such as phrase structure theory and constituency relations are integrated into them to improve their performance.

Keywords: Question Answering, Word-level Question Answering, Sentence-level Question Answering, Neural Network, Knowledge Graph, Convolution Neural Network, Recurrent Neural Network, Constituency Relation

Název práce: Hybridní hluboké metody pro automatické odpovídání na otázky

Autor: Ahmad Aghaebrahimian

Ústav: Ústav Formální a Aplikované Lingvistiky

Vedoucí disertační práce: RNDr. Martin Holub, Ph.D., Ústav Formální a Aplikované Lingvistiky

Abstrakt: Automatické odpovídání na otázky jakožto jedna z nejstarších úloh z oblasti zpracování přirozeného jazyka je jednou z nejzajímavějších a nejnáročnějších oblastí výzkumu s množstvím vědeckých a komerčních uplatnění. Odpovídání na otázky jakožto disciplína se ve spojení s informatikou, statistikou, lingvistikou a kognitivní vědou zabývá tvorbou systémů, které automaticky vyhledávají odpovědi na otázky kladené lidmi v přirozeném jazyce. Tato doktorská disertační práce představuje autorův výzkum uskutečněný v uvedené oblasti. Autor předkládá především své studie a výzkum zaměřený na hybridní systémy pro odpovídání na otázky zahrnující vyhledávací stroje pracující jak se strukturovanými, tak s nestrukturovanými daty. Jádrem strukturovaného vyhledávacího stroje je state-of-the-art systém založený na znalostních grafech. Nestrukturovaný vyhledávací stroj je tvořen state-of-the-art systémem pro odpovídání na otázky na větné úrovni a systémem pro odpovídání na otázky na úrovni slov s výsledky, které se blíží tomu, čeho dosahují lidé. Tato práce představuje také nově vyvinutý standardní soubor "zlatých" dat pro odpovídání na otázky jak na větné úrovni, tak na úrovni slov. Počínaje jednoduchým modelem logistické regrese, autor postupně prezentuje pokročilejší modely včetně řešení pomocí vícevrstvé neuronové sítě, konvoluční neuronové sítě a rekurentní neuronové sítě pro různé výše uvedené systémy pro odpovídání na otázky. Přestože jádrem všech modulů vyvinutých v této práci jsou nově navržené struktury hlubokých neuronových sítí, je do nich také zaintegrováno několik principů založených na jazykové intuici jako např. teorie frázové struktury a vztahy konstituentů, a to za účelem zlepšení jejich výkonnosti.

Klíčová slova: Odpovídání na otázky, odpovídání na otázky na úrovni slov, odpovídání na otázky na úrovni vět, neuronová síť, znalostní graf, konvoluční neuronová síť, rekurentní neuronová síť, vztah konstituentů

Contents

1. Introduction

Question Answering (hereinafter, QA) as a commercially appealing and scientifically challenging research area has fascinated many engineers and scientists from the earliest days of artificial intelligence in the 60's. QA has been even suggested to replace the Turing test as a measure of intelligence (Clark and Etzioni, 2016). It has gained worldwide fame since IBM Watson (Ferrucci et al., 2010) beat two lifelong champions of Jeopardy quiz show in 2011 with a large margin.

Although many significant strides have been made in recent years in this field, QA is still far from being solved. Until recently with some few exceptions (e.g., IBM Watson), QA was mainly limited to some expert systems in limited domains such as air traffic information systems.

Nowadays, however, due to substantial public interest in information access, open-domain QA systems have attracted much more attention. The emergence of big data repositories, textual databases, knowledge graphs, various social networks, and generally the Internet in addition to the introduction of some advanced learning techniques such as Deep Neural Networks (DNN) have all opened up an excellent opportunity for developing open-domain QA systems. These systems have posed a challenge and an opportunity in the field at the same time. They are challenging because inferring the answer of a question among so much data is not a trivial task. They also offer an opportunity, because they pave the way for developing a wide array of useful applications such as dialogue systems, tutoring systems, scientist's assistants, etc.

In recent years, open-domain QA systems succeeded to make use of knowledge graphs efficiently and successfully (Aghaebrahimian and Jurčíček, 2016a; Bordes et al., 2015; Berant et al., 2013). Knowledge graphs (e.g., Freebase (Bollacker et al., 2007)) are large ontologies of concepts and their relations[1]. They are clean, efficient and scalable data structures for entity detection and relation extraction. However, they are sparse, and they lack many entities, and properties[2]. It is because new concepts and properties are always being evolved and any newly compiled knowledge graph today, irrespective of how much comprehensive it is, will be obsolete after passing some relatively short time.

Moreover, knowledge graphs are only able to provide an answer to structured

[1]To read more and get the basics of knowledge graphs, please see Section 2.2.

[2]An entity is a popular thing, place or person. A property is an attribute which is defined by an entity.

or factoid questions. Factoid questions are those questions which ask about an entity. In contrast, unstructured questions are mostly seeking for a non-entity answer such as a definition or any span of consecutive words.

Unstructured[3] QA (Rajpurkar et al., 2016) is a somewhat new challenge in open-domain QA in which the questions are answered by searching through or reasoning over semi-structured or unstructured repositories of data like Wikipedia or the Internet. This type of QA can be defined with different granularities, from sentence-level to phrasal or word-level QA. Compared to structured QA systems based on knowledge graphs, unstructured QA is much more computationally demanding.

To answer structured and unstructured questions by making use of the rich data structure of knowledge graphs and the dynamic and evergrowing structure of Wikipedia, we have developed a hybrid system which consists of two modules: the unstructured module which extracts answers from Wikipedia and the structured module which extracts answers from a knowledge graph. In the next section, we describe our hybrid QA system in more details.

1.1 Hybrid Open-Domain QA

In a broad perspective, QA systems fall into domain-dependent and domain-independent systems.

In domain-dependent systems, the questions are limited to a specific domain like health care. In these systems, the entities (e.g., different drugs, diseases, products, etc.) and properties (e.g., symptoms, side effects, etc.) are limited and already defined.

A domain-independent or open-domain QA system, in contrast, recognizes a non-deterministic number of entities and properties. It requires having access to a significant source of information like knowledge graphs. Some knowledge graphs are big enough to accommodate millions of entities and thousands of properties. However, they have some issues which limit their functionality for open-domain QA.

Knowledge graphs are not dynamic, and due to sparsity inherent in them, they cannot recognize new entities and properties. For instance, consider 'Parenthood' as a typical property in a typical knowledge graph like Freebase. One can query

[3]Free text or non-factoid QA are other names for this type of QA.

'Barack Obama' for his parents, however querying 'Brad Pitt' for his parents returns no answer because the node of 'Brad Pitt' in the Freebase graph has no value for 'Parenthood' property.

In an open-domain system, new entities and properties are evolved continuously. Therefore, the system should be able to get new knowledge steadily. This process should be done automatically and without any human intervention. The other problem with knowledge graphs is that they are not designed to answer non-factoid questions.

One feasible solution for addressing these issues is to use the Internet to support the knowledge graph for its missing facts. The Internet is a dynamic source of information. However, compared to knowledge graph-based models, web-based QA models usually perform sub-optimal in answering factoid questions.

By a hybrid open-domain system (Aghaebrahimian, 2017c), we mean a unified system which is composed of a knowledge graph and a web-based QA models. Integrating these two technologies in a system makes up for the deficiencies of each and makes the final performance superior to the one of each.

In the rest of this section, we enumerate three classes of questions that our hybrid system is designed to answer. Then, we explain the type of reasoning required to answer each class. Finally, we describe the architecture of our system. We use the following passage as a reference context for all the following questions.

On May 21, 2013, NFL owners at their spring meetings in Boston voted and awarded the game to Levi's Stadium. Troy Vincent, the NFL vice President, then went to have a meeting with the authorities of the stadium.

Our hybrid system is designed to answer these three categories of questions.

1. Structured questions with available answers in a knowledge graph:

   Structured questions are answered by querying a knowledge graph using one property and one entity.

   Question: Who is the vice president of NFL?
   Entity: NFL
   Property: Vice President
   Answer: Troy Vincent
   Reasoning: Knowledge graph types and lexical distance

2. Structured questions with non-available answers in a knowledge graph:

   Similar to the questions in the previous group, the answers to these questions are an entity. However, due to sparsity, their answers are not available in a knowledge graph. These questions are answered using unstructured search techniques.

   Question: Who did award the games to Levi's Stadium?
   Property: game_awarded_by
   Entity: Levi's Stadium
   Answer: NFL owners
   Reasoning: Constituency and neural

3. Unstructured questions with available answers on the Internet:

   Unlike the questions in two previous groups, the answers to these questions are a definition, description or generally a span of consecutive words. These questions are answered using unstructured search techniques too.

   Question: What decision did NFL owners make On May 21, 2013?
   Answer: Awarded the game to Levi's Stadium
   Reasoning: Constituency and neural

We explain the components required for answering the first class of questions in Chapter 6. Then we describe the components necessary for answering the other two classes of questions in Chapters 7 and 8 respectively. All the components are depicted in the system architecture in Figure 1.1. In this figure, the components in green are used for doing preprocessing tasks for which we use available off-the-shelf toolkits. They include:

- A syntactic parser and an embedding module: The syntactic parser generates constituency parse trees for the system. We use the Stanford Core NLP toolkit (Manning et al., 2014) in this module. We also use NLTK toolkit to perform some preprocessing and normalization tasks like tokenization, removing stop words, removing punctuation, etc. The embedding module is implemented in three ways; as a look-up table which is trained using available training data, as a pre-trained model using Word2Vec (Mikolov et al., 2011) toolkit and using pre-trained Glove vectors (Pennington et al., 2014a).

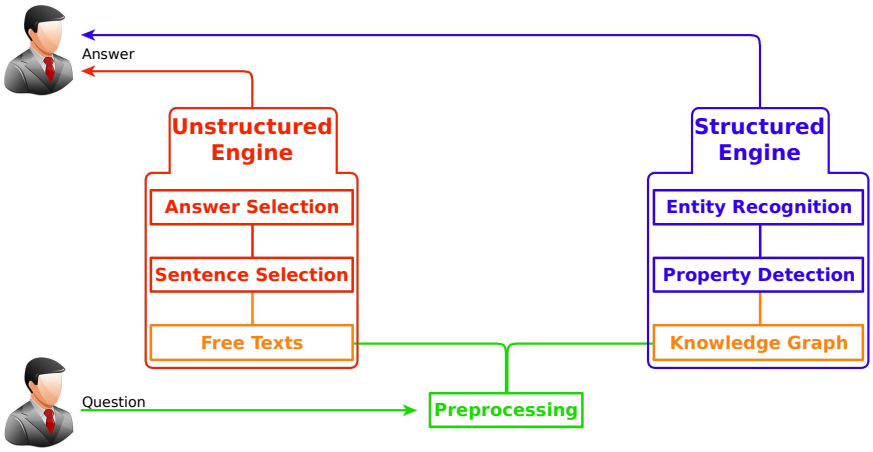


Figure 1.1: General system architecture

The components in orange are the data sources from where the answers are extracted. They include:

- Free texts component which is a corpus of Wikipedia articles included in the SQuAD (Rajpurkar et al., 2016) dataset.
- Knowledge graph which is a copy of the Freebase (data dumps, 2015) loaded into the Virtuoso engine.

The blocks in blue are the components of the structured engine for answering factoid questions. They include:

- Property detection component which returns a distribution over the system's properties given each question.
- Entity recognition component which selects the best matching entity given each question.

The red blocks are the components of the unstructured engine for answering unstructured questions. They include:

- Sentence selection component which selects the best possible answer sentence given a question.
- Answer selection component which selects the shortest best answer from the best-selected sentence.

In the rest of this section, we briefly introduce the structured and unstructured engines. For a more exhaustive description of the system modules, please refer to Chapters 6, 7, and 8 for Structured QA, Sentence Selection, and Unstructured QA modules respectively.

1.1.1 Structured Engine

The structured engine is optimized to answer factoid questions (i.e., questions in the first class). It makes use of two components: property detection and entity recognition.

The property detection component uses different models to estimate a distribution over its knowledge graph properties given each question. It generates a list of n-best property for each question.

Given a list of n-best property and a question, the entity recognition component reasons over all possible entities available in the question to select the best. The reasoning process is two folded; reasoning over the types of the entities[4] and reasoning over the similarity between knowledge graph's entities and question's entities. It computes a score for each possible combination of properties and entities in a given question and returns the highest scored combination as the best answer.

1.1.2 Unstructured Engine

The unstructured engine answers unstructured questions (i.e., the second and the third class of questions explained above). Given a question and a paragraph, the task in unstructured QA is to extract the shortest possible span of words out of the accompanying context (e.g., paragraph). Statistically, we are interested in estimating

$$p(\mathbf{a}|\mathbf{q}, \mathbf{p})$$

where $\mathbf{a}$ is an answer given question $\mathbf{q}$ and paragraph $\mathbf{p}$. Maximizing this

[4]To read more about the knowledge graph typing system, please refer to Fig.2.5 and Section 2.2 for description.

probability returns the most probable answer given each question.

$$a_{best} = argmax_a \quad p(\mathbf{a}|\mathbf{q}, \mathbf{p})$$

We decompose this objective into two objectives; one for sentence selection and the other for answer extraction.

Sentence Selection. Since the search space in a paragraph for searching all possible span of words is too big, in the first step, the Sentence Selection component selects a sentence which contains the answer and limits the search space to the spans available in it.

$$s_{best} = argmax_{s \in \mathbf{p}} \quad p(\mathbf{s}|\mathbf{q})$$

In some applications, we are interested not only in the answer but also in the evidence based on which the question is answered. The best sentence selected by this component is also used as an evidence to support the extracted answer.

Sentence Selection component provides a mean of early error detection as well. If it could not find a sentence with confidence higher than a predefined threshold, it asks the user to rephrase the question. It prevents providing the user with irrelevant or false information.

Answer Selection. Given a question and its best-selected sentence by the Sentence Selection component, the job of the Answer Selection component is to extract the shortest span of consecutive words as the final answer.

$$a_{best} = argmax_{a \in \mathbf{s_{best}}} \quad p(\mathbf{a}|\mathbf{q})$$

In our system, structured questions are provided with a knowledge graph while unstructured questions are equipped with a textual database (e.g., Wikipedia). Therefore, depending on the input data which is provided with each question either one of the unstructured or structured engines retrieves an answer and returns it to the user.

1.2 Objectives and Contributions

The goal of this thesis is to explore and advance state of the art in Question Answering by integrating new deep neural architectures and addressing some of its issues such as scalability, domain dependence, and answer quality.

The core of this thesis consists of the material that the author has already published in the form of several scientific papers (Aghaebrahimian and Jurčíček, 2016a,b; Aghaebrahimian, 2017a,b, 2018a,b) accepted and presented at international scientific conferences including NAACL, COLING, CoNLL, and TSD among others.

The contributions of this work are the following:

- **Property-driven QA:**

  State-Of-The-Art and highly scalable structured QA

  While the space of entities in an open-domain QA system in infinite, the scope of properties is usually much smaller and finite. In this system, we proposed to break the process of Question Answering into two separate processes; one for detecting the property of a question and the other for recognizing the entity from a knowledge graph. With this formulation, our system not only beat the state-of-the-art in simple Question Answering but also scaled up to a knowledge graph with more than fifty million entities while other systems were limited to at most five million entities.

- **Multi-part QA dataset:**

  A small dataset for unstructured sentence- and word-level QA

  This work proposed a new challenge to QA datasets, and that is answering questions with chunks of texts taken from different parts of the accompanying text. This feature adds a new layer of complexity to QA systems for answering the questions whose answers are not limited to only one section of text.

- **Deep Sentence Selection:**

  State-of-the-art unstructured sentence-level QA

  We proposed a neural architecture using which we can enforce textual features (e.g., textual similarity) as a hard constraint at the time of training. We also demonstrated that our architecture is highly effective when integrated with the transfer learning technique.

- **Constituency-based QA:**

  Unstructured word-level QA with results near-to-human performance

  We proposed an architecture which can be trained on linguistic constituents hence generating more eloquent and valid answers and in this way enhancing the overall answer quality of the QA system.

1.3 Organization of the thesis

We organize the remaining of this thesis in the following chapters based on which different groups of readers may decide which chapters they would like to read.

In chapter 2, we describe some essential terminology and techniques that are going to be used frequently in the rest of the thesis. This chapter includes a section on Machine Learning (ML) techniques including, Logistic Regression, standard Neural Networks (NN) as well as Recurrent, and Convolution Neural Networks (RNN, CNN). We include a section in this chapter to describe the structure and characteristics of Freebase (Bollacker et al., 2007). We also enumerate the evaluation metrics which are used in our different experiments and the similarity measures which are used as semantic similarity metrics in further chapters.

In chapter 3 a comprehensive literature review on QA is presented. In this chapter, we describe various models, methods and state-of-the-art systems of QA.

In Chapter 4 we enumerate several datasets which have been used for system evaluation in our work. This chapter is a prelude to the next one in which we introduce our Quora Question Answering Dataset (Q2AD).

Chapter 4 is the last introductory research part of this thesis. The next chapter makes the borderline where the author's own contribution to the field starts.

Chapter 5 describes the process of compiling the Q2AD dataset, a multi-part dataset for unstructured sentence- and word-level QA. We use this dataset to eval-

uate our Sentence Selection system presented in Chapter 7. The content of this chapter is based on a paper (Aghaebrahimian, 2017b) presented and published by the thesis author during his Ph.D. program.

In Chapter 6 we describe our QA system for answering structured questions using knowledge graphs. This chapter is taken from a paper (Aghaebrahimian and Jurčíček, 2016a) which was prepared by the thesis author together with his supervisor and was published at the Human-Computer Question Answering workshop at NAACL 2016. Sections 2.2 and 2.4 and Chapter 4 are prerequisite readings for this chapter.

In Chapter 7 and 8 two deep neural models are introduced for unstructured sentence-level and word-level QA respectively.

Chapter 7 describes a state-of-the-art sentence selection architecture accompanied by the experiments on Q2AD and two other well-studied datasets. The content of this chapter is the shortened text of a paper (Aghaebrahimian, 2017a) which was published at TSD 2017 and was prepared by the thesis author while he was working on his Ph.D. thesis.

Chapter 8 describes a neural model enhanced by constituency relations and optimized for answering unstructured word-level questions. The content of this chapter is taken from a paper (Aghaebrahimian, 2018a) which was written by the thesis author during his Ph.D. study and was published in the proceedings of the CoNLL 2018 conference.

Our respected readers with different interests may choose where to start by checking the reading plan in Figure 1.2.

	Introduction	Machine Learning Basics	Knowledge Graph	Similarity Measures	Evaluation Metrics	Literature Review	Datasets	Q2AD	Structured QA	Sentence-level Answer	Unstructured QA	Conclusion
Interested in QA	✓								✓	✓	✓	✓
Interested in Machine Learning		✓							✓	✓	✓	
Esteemed Reviewers	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Interested in QA Datasets							✓	✓				
Interested New Students	✓								✓	✓	✓	✓
Interested in Skimming	✓											✓

Figure 1.2: The thesis reading plan

Although we have tried to keep the chapters self-contained as much as possible, some of them have some dependencies on other chapters which are illustrated in Figure 1.3.

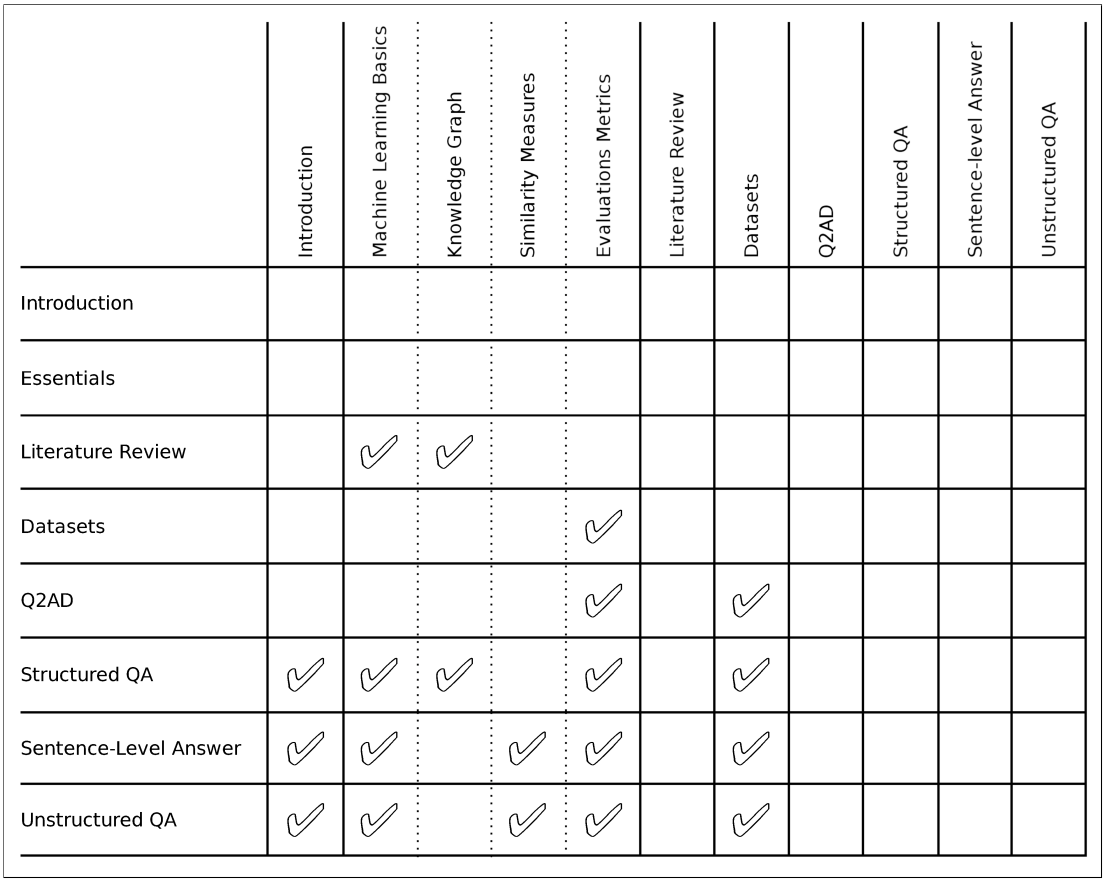


Figure 1.3: The chapter dependencies. The chapters in rows depend on the chapters in columns.

2. Essentials

To make this thesis as self-contained as possible, in this chapter, we try to explain some technical terms and notions that are going to be mentioned in the next sections frequently. We do not intend to make this chapter as a general introduction to Machine Learning. Instead, we only describe particular methods that are used in further chapters. We represent them in the same order as they are used later.

2.1 Machine Learning Basics

2.1.1 Logistic Regression

Logistic Regression is one of the most mathematically elegant and straightforward techniques of supervised machine learning for classification. In supervised machine learning, we have a set of discrete classes, and some samples each labeled with one of the classes. If the number of the classes is less then three, then we deal with a binary classification task. Otherwise, it would be a multi-class classification task.

Inputs to the Logistic Regression algorithm:

$$\text{training data: } (x^1, y^1), (x^2, y^2), ..., (x^m, y^m)$$
$$\text{classes: } y_i \in [y_1, y_2, ..., y_n]$$

The superscripts on the data points above are their index. Therefore we have m training samples. The data points are multidimensional, so we use subscripts to refer to specific dimensions of a data point. For instance, x_2^4 refers to the second dimension of the fourth data point. The labels' subscripts are their index, and their superscripts say to which data points they belong. For instance y_2^5 mentions that the label of the fifth data point is the second class in the set of classes.

As a simple example let's suppose we need to classify some emails into spam and non-spam classes. Then our classes would be $y \in [1, 0]$, 1 for spam or 0 for non-spam. We have a large number of emails each labeled as either 1 or 0.

$$(x^1, 1), (x^2, 1), (x^3, 0), ..., (x^m, 1)$$

For simplicity, we assume the text of each email as a collection of words. Therefore each $x^i \in \mathbb{N}^{|V|}$ is a $|V|$ dimensional vector. $|V|$ is the number of all unique words in the collection of all emails. Each x^i_j mentions how many times the j^{th} word is seen in the i^{th} sample. This technique for text representation is called Bag-of-Words (BOW).

Although BOW is not based on any linguistic theory, it works astonishingly good in many NLP tasks. There are more advanced techniques for text representation like word embeddings which will be discussed in the next section. For now, let's assume we transform our emails into BOW representation in $|V|$ dimensional vectors. We need to decide how much each of the dimensions or features in each email vector is responsible for assigning either 1 or 0 to it. Hence we should learn a new vector called λ which is a $|V|$ dimensional vector again. Learning λ or the parameter vector is the beginning point in machine learning. The goal is to train λ and use it in the linear function

$$y = \lambda^\intercal x + b \tag{2.1}$$

to predict the label of new data points. For more expressiveness of the model, we add a bias term in the function. The output of this function is a real number called logit. To transform logits into probabilities, we use the sigmoid function (Equation 2.2) which converts the logits into a monotonically increasing value from 0 to 1. The output of the sigmoid function is the probability of getting $y = 1$.

$$sig(\lambda^{\intercal}x + b) = \frac{1}{1 + e^{(-\lambda^{\intercal}x+b)}} \tag{2.2}$$

Now we define the objective function which computes the number of errors made by this algorithm. To tune the parameters, we need to know how much each parameter is responsible for the mistakes. To find out this, we need to derivate the sigmoid function with respect to each of the parameters. Unfortunately, the sigmoid function is not convex which means using this function we can not guaranty finding a global minimum for our algorithm.

In simple words, a convex function has only one global minimum. mathematically, a function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if the domain of f is a convex set or formally

$$f(\theta x + (1 - \theta)y) \leqslant \theta f(x) + (1 - \theta)f(y)$$

$$st : 0 \leqslant \theta \leqslant 1, \text{ and for all } x, y \in \mathbb{R}^n$$

To rectify this problem, we use the logarithm of this function and define a two-part objective function.

$$objective = \begin{cases} -\log(f(x)), & \text{if y = 1.} \\ -\log(1 - f(x)), & \text{otherwise.} \end{cases} \tag{2.3}$$

In Equation 2.3, $f(x)$ is the sigmoid function defined in Equation 2.2. These two terms are the cost for one data point. To compute the cost for all data points, we combine these two terms in one equation and compute it for all m samples.

$$objective = \frac{1}{m}\Big[\sum_{i=1}^{m} -y^i \log(f(x^i)) - (1-y^i)\log(1-f(x^i))\Big] \tag{2.4}$$

To improve the generalizability of the model and to prevent it from over-fitting or under-fitting, we add a regularization term to Equation 2.4 and tune it using θ, the regularization term.

$$objective = \frac{1}{m}\Big[\sum_{i=1}^{m} -y^i \log(f(x^i)) - (1-y^i)\log(1-f(x^i))\Big] + \frac{\theta}{2m}\sum_{j}^{n}\lambda_j^2 \tag{2.5}$$

Now we can compute the gradients of the objective function with respect to each of the parameters.

$$\frac{\partial objective}{\partial \lambda_j} = \frac{1}{m}\sum_{i=1}^{m}(f(x^i) - y^i)x_j^i \tag{2.6}$$

In this equation, x_j^i refers to the j^{th} dimension of the i^{th} training sample. Now we can use our labeled training data and an iterative technique called Gradient Descent (GD) to train our parameters.

Gradient Descent

Gradient-based methods are one of the widely used classes of optimizers for minimizing the objective functions in machine learning. Gradient Descent or Batch Gradient Descent as a method of this class is an unconstrained first-order optimization algorithm. The intuition of this method is as follows:

We initialize our parameters with random numbers and compute the gradients $\bigtriangledown$ (i.e., derivative of the objective function with respect to each of the parameters). Then we take a step of length $\eta\bigtriangledown$ on the opposite direction of the gradient to find new parameters.

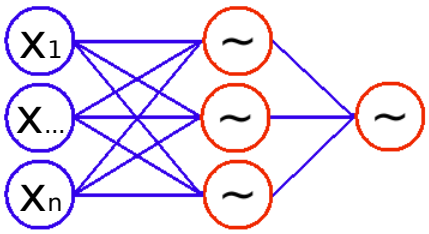


Figure 2.1: A simple Neural Network architecture

$$\lambda_1 = \lambda_0 - \eta \bigtriangledown_{x_0} f(x) \tag{2.7}$$

Then, we compute the gradient at this new point and proceed until we reach a minimum. If we managed to reach the global minimum the norm of the gradient shall be zero (i.e $|| \bigtriangledown f(x) = 0||$). If not, we can define a threshold based on the number of iterations or the value of the norm and stop minimizing after we passed the threshold.

In Batch Gradient Descent for each new step, we need to compute the gradients using all training samples. In large datasets, though, it is very time-consuming. A more practical way is to choose some examples stochastically and compute the gradients of them instead. This method does not guaranty obtaining the global minimum but it is much faster, and usually, the difference between the local minimum obtained by Stochastic Gradient Descent (SGD), and batch Gradient Descent is not noticeable.

2.1.2 Neural Networks

To describe the basics of Neural Networks, we return to our logistic unit in Equation 2.5. A Neural Network is a generalization of logistic units. They feed the output of a linear function to a non-linearity (sigmoid here) and stack them on top of each other to generate complex non-linear functions. Figure 2.1 is a simple Neural Network in which each red circle is a logistic unit. The first and last layers are called the input and output layers respectively, and the inside layer is called the hidden layer. This arrangement helps us to use a simple linear model for building complex non-linear models.

The objective function in Neural Networks is similar to Equation 2.5 with the difference that the objective here should be computed for each neuron. It means that in addition to x which is parametrized for each sample, y is also parametrized for each layer. Moreover, in the regularization term, each parameter itself is parametrized by previous and next layer.

$$\begin{aligned} objective = & \frac{1}{m}\Big[\sum_{k=1}^{K}\sum_{i=1}^{m} -y_k^i \log(f(x^i)_k) - (1-y_k^i)\log(1-f(x^i)_k)\Big] + \\ & \frac{\theta}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{n_{s+1}}(\Lambda_{ji}^l)^2 \end{aligned} \tag{2.8}$$

In Equation 2.8, s is the number of neurons in layer l, Λ is the matrix of all parameters for all neurons in all layers, K is the number of neurons in the last layer (i.e., number of classes) and θ is the regularization term.

2.1.3 Convolution Neural Networks

Convolution Neural Networks (CNN) (LeCun et al., 1998) is one of the widely recognized variants of Deep Neural Networks (DNN). A DNN is a neural network with a large number of layers as the hidden layer. CNNs are mostly recognized for their influence on image recognition. However, we will see that they can be beneficial in NLP too.

Convolution is a feature extraction mechanism which can be thought of as a sliding window applied to a design matrix. The design matrix is a matrix in which the rows are tokens in sentences typically a word or even characters, and the columns are features of the symbols. These features can be one-hot vectors indexing words in a vocabulary set or pre-trained word embeddings like Word2Vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014a). In a real application, there are hundreds of such convolutions known as filters, kernels or feature detectors of different sizes. There are wights associated with each cell in filters which are learned through training. For more elaboration, we assume we have a sentence with five tokens where each token has a 5-dimensional vector. Then we have a 5*5 dimensional design matrix.

In contrast to image recognition applications, in NLP, convolution filters are

slid only vertically, and their width is always the same as the number of columns in the design matrix. Their length, however, can be variable and is usually between one and seven. In a typical CNN, we can have several filters each with different lengths.

$$conv = nonlinearity(W^{\intercal}x + b) \tag{2.9}$$

With each cell of these filters, a parameter (i.e. weight) is associated. These filters slide over the design matrix vertically with various step-sizes. A nonlinearity function like Hyperbolic tangent (tanh)(Equation 2.10) or Rectified Linear Unit(ReLU)(Equation 2.11)

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.10}$$

$$ReLU(x) = x^{+} = \max(0, x) \tag{2.11}$$

is applied to the linear combination of their weights and the features. Each filter outputs a real number and sliding these filters with different length over the design matrix generates feature vectors with different lengths. To create a fix length vector usually a max-pooling or average-pooling layer is applied after the convolution layer to extract the max or average values out of all extracted convoluted vectors. The pooling mechanism not only fixates the vector size which is a must for the classification task but also reduces the dimensionality of the input by retaining the most salient information in vectors. In the end, a softmax function (Equation 2.12) is applied to the max or average-pooled vectors to determine the class.

For more illustration, we consider the design matrix above with 5*5 dimensions. We assume that we have three 3*5 filters. Then by sliding the filters on our design matrix, we will have three 3-dimensional vectors. If we max pool these vectors we will have a 3-dimensional vector which can be fed into a softmax function. Figure 2.2 illustrates a simple CNN with the components described above.

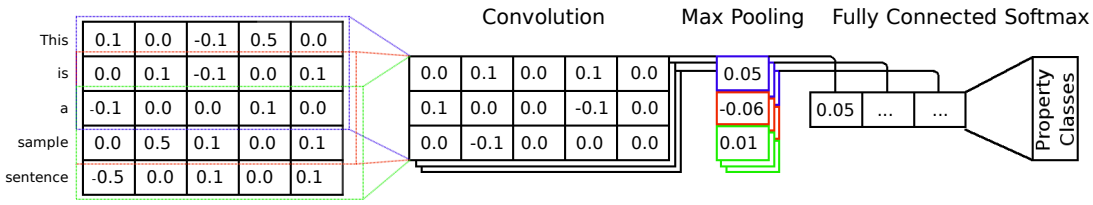


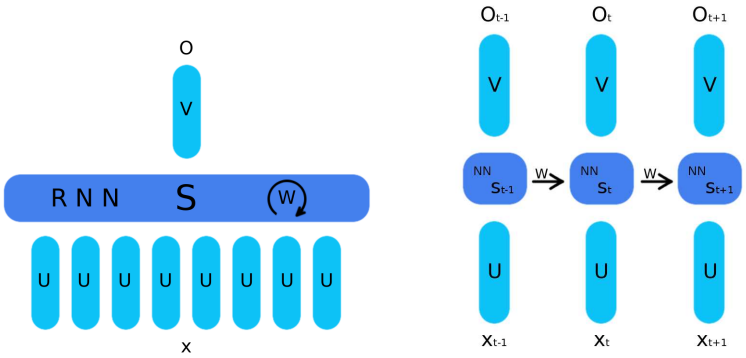
Figure 2.2: A CNN with three convolution filters, Max pooling layer, and a fully connected softmax

$$softmax(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{|\mathbf{x}|} e^{x_j}} \tag{2.12}$$

Compared to the n-gram approach in which all possible n-ary combinations of adjacent words are considered, convolutions learn word representations very efficiently. Computing word representations more than tri-grams especially when we are dealing with large vocabularies is very expensive and not efficient. However, convolution filters with different length represent words very similar to the way n-grams do but in a very compact and efficient form.

2.1.4 Recurrent Neural Network

Recurrent Neural Network (RNN) (Elman, 1990) is another variant of DNNs. Compared to CNN, Recurrent Neural Networks are more intuitive when we think about language as a left to right or right to left processing task. For a sequence of length n, an RNN repeats itself n times by receiving the output of the previous layer as the input of the current layer.



(a) RNN layers (b) An unrolled RNN

Figure 2.3: a. A regular RNN unrolled in b. U, V, and W are input, output, and internal RNN weight matrices. An RNN cell repeats itself for each token in a sentence.

In simple Logistic Regression or Neural Network models based on BOW, we assume that each word is independent of the others. It is a wrong assumption. RNN is a sequential model which considers the dependence of each word on its previous history (words). RNN persists the information across steps by looping over tokens in a sequence.

As depicted in Figure 2.3 an RNN receives the input as x_t to compute s_t, the hidden state of the RNN in step t:

$$s_t = nonlinearity(U^\intercal x_t + W^\intercal s_{t-1}) \tag{2.13}$$

The nonlinearity is usually tanh or relu. The output is computed using softmax of h_t over all possible classes.

$$o_t = softmax(V^\intercal s_t) \tag{2.14}$$

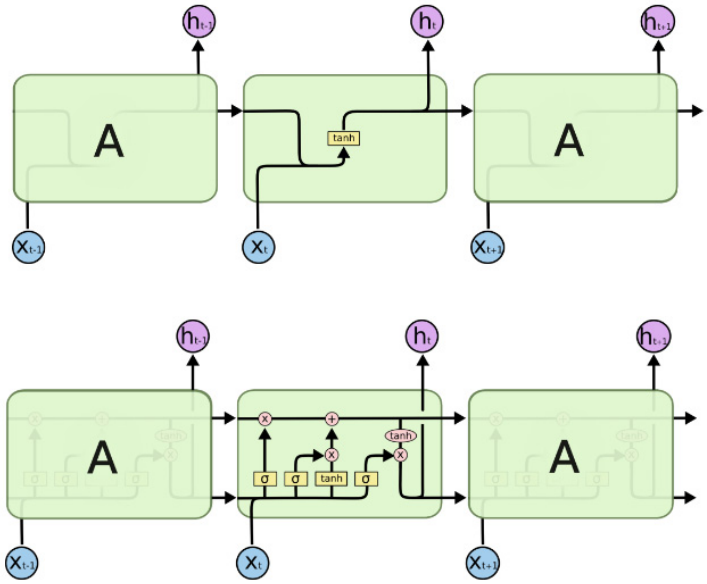


Figure 2.4: An LSTM with the internal gates[1].

Theoretically, an RNN can memorize the information of an arbitrarily long sequence. However, since the parameters (i.e., U, W, V) in RNNs are shared across all steps to decrease the total number of parameters, they are only capable of learning short-distance dependencies.

RNN's parameters are trained using backpropagation, the same technique used for training simple and Convolution Neural Networks. However, since the weights in RNN layers are shared, at each step gradient computations should be done for all steps. It means if we are in step five we need to backpropagate and sum up all gradients until step one. This is called backpropagation through time (BPTT).

Due to the vanishing/exploding gradient problem caused by BPTT, very long-distance dependencies are not detectable in regular RNNs. Therefore we resort to a variation of RNN called Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) which is specifically designed to address this problem.

The primary distinction that separates LSTMs from vanilla versions RNNs is their capacity to learn long-distance dependencies. LSTMs make use of previous history h_{t-1}, current input x_t, and current memory s_t to decide what to add to, what to keep in and what to forget from memory.

As shown in Figure 2.4, an LSTM cell consists of four neural networks (compared to a regular RNN which includes one) arranged in a specific architecture.

[1]Image courtesy: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

C_t is cell state in step t. LSTMs can manipulate their cell state by adding or removing information to/from it. They do it using specific structures called gates. Gates consist of a sigmoid nonlinearity and a pointwise multiplication operation. The output of sigmoid is between 0 and 1, and when multiplied by a feature, it determines how much of it should be retained and how much should be forgotten. Zero means to forget all and one means to retain all. An LSTM has three gates; the forget, the input and the output gate.

The first gate is the forget gate f_t. For each feature in C_t the forget gate generates a real number between zero and one to decide how much of them should be retained and how much should be thrown away.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{2.15}$$

The next gate, input gate, decides what information should be added to cell state. It does it in two steps. First, it generates a vector of data that could be added using a tanh nonlinearity n_t. Then a sigmoid nonlinearity generates a real number i_t for each feature in n_t to determine how much of each should be added to the cell state. The result of pointwise multiplication between i and n is then added to the cell state.

$$n_t = \tanh(W_n.[h_{t-1}, x_t] + b_c) \tag{2.16}$$

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{2.17}$$

Before we continue with the output state, we can update the cell state c_t by the vectors generated by the forget and input gates. It is done by scaling the cell state in the previous step c_{t-1} by point-wise multiplying it with the forget vector and then to add the result to the scaled input vector.

$$c_t = f_t * c_{t-1} + i_t * n_t \tag{2.18}$$

Finally, the output gate decides what information the LSTM cell should output. For doing so, in the first step, a tanh nonlinearity is applied to the cell state. The resulting vector is then scaled using a sigmoid nonlinearity which is applied to the concatenation of previous cell output h_{t-1} and current input x_t.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{2.19}$$

$$h_t = o_t * \tanh(c_t) \tag{2.20}$$

In the end, A softmax of h_t over all possible classes determine the most probable class given each sample.

2.2 Knowledge Graph

Knowledge graphs contain significant quantities of factual information about entities and their attributes such as Place of birth or Profession. Large knowledge graphs generally cover numerous domains, and they may be a solution for scaling up domain-dependent systems to open-domain ones by expanding their boundary of entity and property recognition. Besides, knowledge graphs are instances of linked-data technologies. In other words, they can be connected easily to any different knowledge graph, and it increases their domain of recognition. In this section, we describe the structure of the Freebase knowledge graph (Bollacker et al., 2007) which is used in this thesis for answering structured question.

Like ontologies, knowledge graphs like the Freebase or Wikipedia are knowledge bases of entities and their relationships. Structurally, a knowledge graph can be described as a graph of entities where some links connect the entities to each other.

A unique characteristic of the Freebase is that the links are labeled to describe the relationship between two entities at two ends of each link. Since the links are bidirectional, they have two labels for each direction. The other unique characteristic of the Freebase is that every entity in it has two unique identifiers which are reliable ways for referencing. These identifiers are *'ID'* and *'MID'* which are human-readable and machine-readable codes respectively.

The entities in knowledge graphs do not have any meaning by themselves. These are the properties connected to them which create the meaning. For instance an entity with *'MID = m/02cft'* does not mean anything until one follows the links connected to it like *'name = Dublin'* etc. Therefore every meaning in the graph is represented by a link.

Beside entity specific properties such as Place of birth for people entities or Capital for country entities, *'ID, MID, name, type'*, and *'expected type'* are some other links which tell essential information about each entity and property.

'name' is a surface form and is usually a literal string in the form of raw text, date or a numerical value. Each entity has a set of *'types'* which describe what kind of entity it is. For instance, an entity like Dublin may have types like Location, Genre, etc. Each link (a.k.a property) has zero or at most one *'expected type'* which predicts the type of the target entity. For instance, *'time_zone'* is the expected type of *'/time_zones'* property[2].

The data in a knowledge graph is arranged in a specific structure called assertion. An assertion[3] is a small labeled, directed graph structure in which a subject entity is connected to an object entity. The connection is made by a link which is labeled by an attribute about the subject. For instance, in Figure 2.5, 'm/02cft' as the subject is connected to 'm/5ghi' as the object. 'time_zones' property makes this connection.

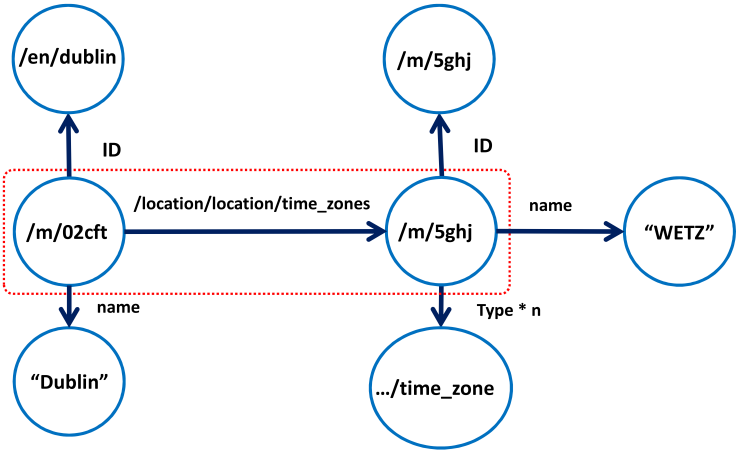


Figure 2.5: The Freebase graph structure

Big knowledge graphs like the Freebase or Wikipedia cover tens of domains and contains millions of entities. As of winter 2015, the Freebase contained over

[2]Properties are in abbreviated form to save space. For instance, the full form for this property is *'location/location/time_zones'*

[3]Please see the dotted line in Figure 2.5

50 million entities and over 500 million links with over 14K different types in more than 80 domains.

The Freebase is accessed whether using the Freebase API provided by Google using Metaweb Query Language (MQL)[4] or using SQARQL query language on its dump data. The Freebase integrated into Google knowledge graph in 2015, and since then, it can be accessed using the Google API dedicated to this knowledge graph. The following queries are MQL and SQARQL equivalent queries for retrieving the name of the entity with *'MID = m/02cft'*:

```
// MQL
{"mid":"m/02cft",
       "name":null}

// SPARQL
PREFIX fb: <http://freebase.com/>
SELECT ?name
WHERE {
    ?mid fb:mid "m/02cft" .
    ?mid fb:name ?name .
}
```

Listing 2.1: MQL and SPARQL query for name retrieving. http://freebase.com/ is the URI defined in a SPARQL triple store engine like the virtuoso. It is not a valid URL address.

2.3 Similarity Measures

The first step in performing machine learning algorithms on textual strings is to transform them into vector space. The simplest way to do so is to translate them into one-hot vectors. One-hot vectors are $|V|$ dimensional vectors where each dimension is an element of the vocabulary. A One-hot vector transforms a string into a point and place it in a $|V|$ dimensional space where each word in the vocabulary is an axis.

One-hot representation is a naive and sparse representation. Since each word is considered as a discrete entity, the semantic similarities among the words are

[4]MQL is a template-based querying language which uses Google API service for querying the Freebase in real time.

disregarded. One approach to improve this deficiency is to include bigrams, trigrams, and more combined units of vocabularies as the separate axis in the space of vocabularies. However, by expanding the size of the vocabulary, the size of the vectors in this method increases with no boundaries which makes it an inefficient representation for applications with open-class vocabularies.

The solution to this problem is to use RNNs to learn low-dimensional word vectors called word embeddings (Mikolov et al., 2013) and use them instead of words.

When textual strings are transformed into vector space either using one-hot vectors or word embeddings they are basically some points in a multidimensional space hence the distance or similarity among them can be computed mathematically.

The first choice for distance estimation is the Euclidean distance. Euclidean distance is the straight-line distance between two points in Euclidean space. However, Euclidean vectors for documents with different length is large. Let's imagine two very similar vectors in a vector space. The distance between the two vectors is quite large because the angle between them is tiny. Therefore, the angle between two vectors is a better representative of their semantic contents. In this way, when the similarity between two sentences increases, the angle between their vectors decreases and vice versa.

Since the angle is a discrete value, we use the Cosine of the angle which is a monotonically decreasing function between 0-180 degrees. Therefore, dropping the Cosine is the same as increasing the angle and hence decreasing similarity. In other words, the higher the Cosine between two vectors the more similar they are. Before we compute the Cosine, we need to make the vectors' length normalized. Length normalization puts the vectors on the surface of the unit hypersphere. It is done by dividing the elements of the vectors by its length. For this purpose, we can use L2 norm function (Please see Equation 2.21).

$$\|x\|_2 = \sqrt{\sum_i x_i^2} \tag{2.21}$$

When the length of vectors is normalized, the Cosine similarity between them is defined as the dot product or sum over element-wise multiplication between two vectors.

$$\cos(\vec{\boldsymbol{v_1}}, \vec{\boldsymbol{v_2}}) = \vec{\boldsymbol{v_1}}.\vec{\boldsymbol{v_2}} = \sqrt{\sum_{i}^{|V|} v_{1_i} \cdot v_{2_i}} \tag{2.22}$$

A full survey of lexical association measures is presented in Pecina (2008). For the sake of completeness, we enumerate some of the most common vector space similarity functions according to Feng et al. (2015b).

- Polynomial:

  $(\vec{\boldsymbol{v_1}}, \vec{\boldsymbol{v_2}}) = (\gamma \vec{\boldsymbol{v_1}}.\vec{\boldsymbol{v_2}} + c)^d$

- Sigmoid:

  $(\vec{\boldsymbol{v_1}}, \vec{\boldsymbol{v_2}}) = \tanh(\gamma * \vec{\boldsymbol{v_1}}.\vec{\boldsymbol{v_2}} + c)$

- Radial Basis:

  $(\vec{\boldsymbol{v_1}}, \vec{\boldsymbol{s}}) = \exp(-\gamma \left\| \vec{\boldsymbol{v_1}} - \vec{\boldsymbol{v_2}} \right\|^2)$

- Euclidean:

  $(\vec{\boldsymbol{v_1}}, \vec{\boldsymbol{v_2}}) = \frac{1}{1+\|\vec{\boldsymbol{v_1}} - \vec{\boldsymbol{v_2}}\|}$

- Exponential:

  $(\vec{\boldsymbol{v_1}}, \vec{\boldsymbol{v_2}}) = \exp(-\gamma \left\| \vec{\boldsymbol{v_1}} - \vec{\boldsymbol{v_2}} \right\|)$

- The Geometric mean of Euclidean and Sigmoid Dot product:

  $(GESD)(\vec{\boldsymbol{v_1}}, \vec{\boldsymbol{v_2}}) = \frac{1}{1+\|\vec{\boldsymbol{v_1}} - \vec{\boldsymbol{v_2}}\|} * \frac{1}{1+\exp(-\gamma(\vec{\boldsymbol{v_1}}.\vec{\boldsymbol{v_2}}+c))}$

The hyperparameters in all cases including γ, c, and d are determined using validation data.

2.4 Evaluation Metrics

The performance of a QA system can be evaluated using several statistical measures depending on the task which is accomplished in the system. In this section, we describe exact match Accuracy, F1 (Macro-averaged), Path Accuracy, Mean Average Precision (MAP), Mean Reciprocal rank (MRR), BLEU and ROUGE as various evaluation metrics which we used in different experiments in this work.

In a structured QA system where the answer is an entity, the answer is either true or false. Therefore, the Accuracy is the most convenient and informative metric for system evaluation.

$$\text{Accuracy} = \frac{|\text{correct answers}|}{|\text{questions}|} \tag{2.23}$$

In QA systems based on knowledge graphs, the answer extraction process is a graph traversal problem where to find an answer, one needs to know the correct source entity and the correct property. So in such systems, an answer is considered correct only if the predicted entity and property are both correct. In these systems, we use the Path Accuracy.

$$\text{Path Accuracy} = \frac{|\text{answers with correct entity AND property}|}{|\text{questions}|} \tag{2.24}$$

The Accuracy is used when there is only one ground-truth answer, and the predicted response can be either the same as or different from it. However, when the ground-truth answers consist of more than one part, we need to use F1 score which is the harmonic mean of the Precision and the Recall.

$$\text{Precision} = \frac{|\text{predicted correct answers}|}{|\text{all correct answers}|} \tag{2.25}$$

$$\text{Recall} = \frac{|\text{predicted correct answers}|}{|\text{all answers}|} \tag{2.26}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.27}$$

Macro-averaged F1 is used to evaluate QA systems in which the predicted answer as a span of words can be a subpart of ground-truth responses. This metric measures the average overlap between a prediction and the ground-truth answer. For each question, the prediction and the ground-truth answer are treated like bags of tokens on which their F1 is computed. The average of all calculated F1 for all questions is then reported as the Macro-averaged F1.

In all the measures mentioned above, we assume that the QA system predicts only one answer. In some QA systems, the prediction consists of a ranked list of answers. In such systems, it is desirable to consider the order in which the predicted responses are presented. The performance of these systems is measured using the mean of the average precision (MAP) and the mean of the reciprocal ranks (MRR).

The MAP is the mean of the average precision of all questions in a test dataset. Average precision of a question is defined as the sum of precision of predicated answers each in k^{th} cut-off over the number of retrieved answers.

$$\text{Average Precision} = \frac{\sum_{k=1}^{n} precision(k) \cdot id(k)}{|\text{all correct answers}|} \tag{2.28}$$

In Equation 2.28 k is the rank in the sequence of retrieved answers, n is the number of extracted answers, precision(k) is the precision at cut-off k in the list, and id(k) is an indicator function equaling one if the item at rank k is a correct answer, zero otherwise.

MRR is the mean of the reciprocal ranks of all questions in a test dataset. Reciprocal rank is another statistical measure for evaluating QA systems that predict a list of possible answers to a question, ordered by their probability of correctness. The reciprocal rank of the answers to a question is the multiplicative inverse of the rank of the first correct answer.

$$\text{Reciprocal Rank} = \frac{1}{|\text{q}|}\sum_{i=1}^{|q|}\frac{1}{rank_i} \tag{2.29}$$

In Equation 2.29, $rank_i$ refers to the rank position of the first correct answer for the i^{th} question and $|q|$ is the number of questions.

BLEU(bilingual evaluation understudy) (Papineni et al., 2002) and ROUGE (Recall-Oriented Understudy for Gisting Evaluation) (Lin, 2004) are metrics for assessing the quality of an automatically extracted or generated text against a set of references (human-produced).

BLEU compares the number of common n-grams (uni-grams through four-grams) between the extracted text and the references and penalizes the result with the length of the text. ROUGE has several variations like ROUGE-L (Longest Common Subsequence) which identifies the longest co-occurring n-gram sequence. It also takes sentence level structure similarity into account.

3. An Overview of Question Answering Systems

Natural Language Interface to Database (NLIDB) and Machine Comprehension (MC) are two main areas of research which have primarily contributed to the development of QA systems. In an attempt to place our QA system detailed in Section 1.1 in a proper context, we will explain 1) NLIDB systems, 2) cloze-type QA, 3) non-factoid and, finally 4) factoid QA systems[1]. Then, we will have a review on some conventional methods for QA in Section 3.1 and finally, we will elaborate on standard architectures of QA systems in Section 3.2.

NLIDB was the earliest instance of domain-dependent QA systems. Baseball (Green et al., 1961) and Lunar (Woods, 1973) were among the first NLIDB systems.

Baseball was designed to answer questions about American baseball league, and Lunar was a scientist's assistant which was able to answer questions about geology. They were intended to help people communicate with databases in a natural language. They used sophisticated language knowledge to translate users' questions into a standard database query. Although the syntactic parsing capacity of these systems was primarily limited, they were able to answer complex questions.

NLIDB systems did not flourish much in the 80s. One primary reason for this decline was the complexity of the natural language which was used in these systems. Their interface language was not much more straightforward than the database query language itself. According to Androutsopoulos et al. (1995), they even diminished almost entirely in the 90s[2]. However, after the 90s, the advent of some efficient semantic parsing approaches and the possibility of using natural language without any restriction helped to revive QA in some limited-domain applications such as geographical or public transportation QA systems. The ARPA sponsored project, ATIS in air traffic domain and many other projects in restaurant domain flourished at that time.

Compared to NLIDB systems, use of QA as a measure of Machine Compre-

[1]There are other types of systems such as QA for solving mathematical problems (Liguda and Pfeiffer, 2011) or QA for answering questions which require complex reasoning (Khashabi et al., 2016). However, these systems are domain-dependent hence are not in the focus of this work.

[2]The reasons for the decline of NLIDB systems are elaborated in Copestake and Jones (1989).

hension (MC) (Kadlec et al., 2016; Hermann et al., 2015) is rather a new research area. The primary goal of these systems is to make a machine read a text and then answer some multiple-answer (cloze-type) questions.

The first study in statistical MC dates back to Hirschman et al. (1999) who devised the first data-driven approach for QA on a small dataset. The dataset included 600 elementary-school-level reading comprehension questions. Long before that though, in the 60s, Philips A.V. developed Routine (Phillips, 1960) as the first rule-based open-domain QA system for comprehension tests.

Routine is designed to read a text and to answer simple reading comprehension questions. The system categorizes the tokens in a passage into some subject, object, verb, place and time expressions. Then, it analyzes each question based on the training data to decide what expression is the best match for it.

Yang et al. (1999) used QA for MC in an open-domain context too. In his approach, in the first step, the best sentence which possibly contains the answer to a given question is detected. Then, the actual answer can be extracted from this sentence using factor graph on multiple sentences (Sun et al., 2013), dependency trees (Shen and Klakow, 2006), or bootstrapping surface patterns (Ravichandran and Hovy, 2002).

Answer Selection is another form of QA which can be used as a measure of MC (Kadlec et al., 2016; Hermann et al., 2015). In this setting, a typical QA system reads a text and then answers either multiple-answer (cloze-type) or free-text questions. Cloze-type answers are limited to multiple distinct entities (usually 4 or 5) while a span of words answers free-text questions.

The level of understanding in cloze-type questions is variable depending on the type of missing elements. The missing elements in these systems can be a simple word or a phrase or even a full sentence. The possibility of using different types of missing elements makes different levels of difficulty in these systems (Hill et al., 2015). For instance, due to cohesion inherent in original texts, finding prepositions is relatively easy, while looking for named entities is more difficult.

Single-word QA algorithms usually are tested on cloze-type datasets. Cloze-type QA systems are word-level systems where the system is trained to select the answer among four or five answer choices. There are a wide range of datasets for cloze-type (Hermann et al., 2015; Hill et al., 2015) QA. CBT (Hill et al., 2015) and CNN (Hermann et al., 2015) are among the most significant cloze-type datasets which contain more than half a million and one and half a million questions respectively. Cloze-type QA datasets are easy to compile because they

can be generated automatically.

Cloze-type QA provides a reliable measure for MC. However, in QA systems for other purposes like the dialogue systems or scientist's assistants, the answers are not known in advance. It is assumed that they are available somewhere on the Internet. Free-texts searching is a reasonable approach to these systems. In contrast to cloze-type questions, in free-texts or non-factoid QA systems (Rajpurkar et al., 2016), the answers, their boundaries and their types (e.g., proper noun, adjective, noun phrase, etc.) are not known in advance, and it makes this type of QA more challenging. In this setting, free-text QA or QA over unstructured data (Rajpurkar et al., 2016; Cui et al., 2016) is advocated where answers are spans of multiple consecutive words in large repositories of textual data like Wikipedia.

Unstructured QA in recent years has been studied with a few distinguishable different settings such as Answer Selection, Answer Trigging, and Answer Extraction.

In Answer Selection (Aghaebrahimian, 2017a; Wang et al., 2016; Yu et al., 2014) and Answer Trigging (Jurczyk et al., 2016) the goal is to find the best answer sentence given each question. These answer sentences may be non-existent in the provided context for Answer Triggering. In Answer Extraction (Shen and Klakow, 2015; Sultan et al., 2016) a chunk of a sentence as the shortest possible answer is extracted.

Many successful studies have been performed for free-text (i.e., phrase) answer extraction from SQuAD (Rajpurkar et al., 2016) since its release in 2016. Almost all of these models benefited from a form of DNN architecture and a majority of them integrated a kind of the attention mechanism (Seo et al., 2016). Some of these studies incorporated attention to predicting the physical location of answers (Xiong et al., 2016; Cui et al., 2016; Hu et al., 2017). Others made an effort to find a match between queries and their contexts (Cui et al., 2016) or to compute a global distribution over the tokens in the context given a query (Wang and Jiang, 2016). Still, some other models integrated other mechanisms like memory networks (Pan et al., 2017), reinforcement learning (Shen et al., 2016) or constituency relations (Aghaebrahimian, 2018a) to enhance their attention performance.

The last group of QA systems in our review are factoid or simple QA systems (Bordes et al., 2015; Aghaebrahimian and Jurčíček, 2016a,b) like air traffic information (ATIS) or the dialogue systems. Simple QA is a factual retrieval

technique which attempts to find an answer entity in a structured data repository like a knowledge graph. Simple in this context does not mean that it is a simple task. It refers to the fact that answering these questions requires knowing just one property and one entity. Although the systems in this category scale well to big knowledge graphs (e.g., Freebase (Bollacker et al., 2007)), they suffer from sparsity which makes them unable to answer some questions.

The dialogue system is another research area which has helped the evolution of QA systems too. ELIZA (Weizenbaum, 1966) is one of the first QA dialogue systems which uses a sequence of pattern matching and string replacement to make a conversation with patients. ATIS (1989-1995), Verbmobile (1996-2000), How may I help you? (2001), AMITIES (2001-2004), TALK (2009-2011), Classic (2008-2011), Parlance (2011-2014), Carnegie Mellon Communicator (1999-2002), Companions (2006-2010) and Alex (2012-2016) are among other dialogue systems each of which specialized in answering questions in a specific domain.

3.1 Question Answering: Methods

All possible QA methods can be roughly organized into three broad categories; Semantic Parsing (SP), Information Retrieval (IR)/Information Extraction (IE) and most recently, end-to-end Neural Networks (NN).

In QA systems based on SP (Clarke et al., 2010; Kwiatkowski et al., 2010; Wong and Mooney, 2007; Zettlemoyer and Collins, 2005; Zelle and Mooney, 1996) natural language utterances are translated into a logical representation of their meaning in a knowledge representation language like lambda calculus expressions (Carpenter, 1997), lambda-Dependency Compositional Semantics (Liang, 2013), robot controller language (Matuszek et al., 2012), etc.

In this approach, a standard procedure is to over-generate possible meaning representation candidates out of a limited number of predefined entities and properties (i.e., the lexicon). Then, these meaning representations are rated using different possible machine learning methods[3]. Eventually, the highest rated meaning representation is queried against a relevant database to fetch the answer. The domain of available entities and properties in the lexicon of these systems is usually limited and small.

In domain-specific QA, this limitation is handled using a static lexicon for

[3]Please see (Aghaebrahimian and Jurčíček, 2015a) for a review of machine learning approaches for semantic parsing.

mapping surface forms of the entities to their logical forms (Clarke et al., 2010; Kwiatkowski et al., 2010; Wong and Mooney, 2007; Zettlemoyer and Collins, 2005; Zelle and Mooney, 1996). However, scaling up such limited lexicons which usually contain from hundreds to several thousand entities is neither easy nor efficient in open-domain QA. This limitation makes it difficult for QA systems to scale well to open-domain or even to a large ontology. Knowledge graphs Instead, contain millions of entities and are highly efficient structures which can be used for entity recognition.

Knowledge graphs provide rich databases of factual information on well-known people, things and places and they proved to be beneficial for different tasks in NLP including Question Answering. Using big knowledge graphs like Freebase or Wikipedia has been shown beneficial (Berant et al., 2013; Cai and Yates, 2013; Kwiatkowski et al., 2013; Berant and Liang, 2014; Bordes et al., 2015; Aghaebrahimian and Jurčíček, 2016a,b) for expanding the scope of language understanding in such cases. We briefly explained knowledge graphs and their structure in Section 2.2.

There are many studies on using knowledge graphs for QA either through an Information Retrieval approach (Yao and Durme, 2014; Bordes et al., 2015) or Semantic Parsing (Berant et al., 2013; Berant and Liang, 2014; Cai and Yates, 2013; Kwiatkowski et al., 2013). Even in these studies, there is still a list of predefined lexicons for entity recognition (Berant et al., 2013; Cai and Yates, 2013). Essentially, they use knowledge graphs only for validating their generated logical forms and for entity recognition they still depend on some initial lexicons.

The most recent work and state-of-the-art on QA in knowledge graphs belongs to Bordes et al. (2015) in which they used memory networks for answering the questions in SimpleQuestions dataset (Bordes et al., 2015). They tested their system on two limited sub-graphs of Freebase containing two and five million entities (FB2M, FB5M). A 0.5 % decrease in the performance of their system when scaling from FB2M to FB5M suggests that QA in a full knowledge graph is quite a difficult task.

The problem with QA systems based on knowledge graph is that their data in the best case is limited to their knowledge graph beyond which, they are not able to recognize any new entity or property. Besides, in the best case, they are only able to answer factoid questions. Hence, knowledge graphs should have the capacity of expansion if they are supposed to be used in an open-domain QA system.

Vast amount of works are devoted to knowledge graph expansion by adding new information which are extracted by parsing external textual corpora (Suchanek et al., 2007; Socher et al., 2013; Fader et al., 2011; Snow et al., 2005). However, arbitrarily adding data to knowledge graphs does not guarantee an effective data expansion procedure and it does not scale well through time. A more efficient and scalable way for knowledge graph expansion is to resort to an unlimited source of information like the Internet.

Use of knowledge graphs is popular in IR/IE based systems too. In contrast to SP-based QA systems, IR/IE-based systems (Yao and Durme, 2014; Bordes et al., 2015) directly retrieve or extract the answer from a database using different statistical methods. These methods range from simple techniques like Point-wise Mutual Information(PMI) (Church and Hanks, 1990) to more complex ones like Deep Neural Networks (Aghaebrahimian and Jurčíček, 2016b; Dai et al., 2016).

Beside SP and IR/IE based systems, there are some other less-popular varieties like entailment-based systems (Bentivogli et al., 2008) or ensemble models (Clark et al., 2016) which are mostly proposed for domain-dependent QA systems.

In the entailment approach, the system assumes that the entailment 'question+answer' is available in the corpus and tries to detect the answer by finding the corresponding entailment. In ensemble models, an array of solvers each with specific inference algorithm is utilized to infer the answer. These algorithms may range from statistical IR to rule-based or constraint optimization solutions.

Constraint optimization is another active research area in QA in both SP and IE approaches. Constraint optimization using Integer Linear Programming (ILP) has been demonstrated being useful and productive in several NLP tasks (Chang et al., 2012; Srikumar and Roth, 2011; Goldwasser and Roth, 2011; Chang et al., 2010; Roth and Yih, 2004). Especially, it has been reported to obtain state-of-the-art results for answering questions which require complex scientific reasoning (Khashabi et al., 2016).

Constraints are essentially Boolean inequalities which are defined based on some prior linguistic information (e.g., types of entities or properties). Prior knowledge about linguistic structures plays a crucial role in structure prediction especially, where there is not enough annotated training data or when models are too simple to detect long dependencies (Chang et al., 2012).

The last common method of QA is end-to-end neural network (NN) systems. QA in the context of NN is addressed mostly as a sequences prediction or classi-

fication/ranking problem.

A group of NN architectures called Deep Neural Networks (DNN) are mainly known for their superior performance in many NLP tasks including QA. DNNs stack a large number of layers in a Neural Network to extract different levels of representation. In recent years, QA has been largely benefited from the development of DNN architectures largely in the form of Convolution Neural Networks (CNN) (LeCun et al., 1998) or Recurrent Neural Networks (RNN) (Elman, 1990). Neural tensor networks (Socher et al., 2013), recursive neural networks (Iyyer et al., 2014), CNN-based models (Yin et al., 2015a; Dong et al., 2015; Yih et al., 2014), attention models (Hermann et al., 2015; Yin et al., 2015a; Santos et al., 2014) and memory networks (Graves et al., 2014; Weston et al., 2015; Kumar et al., 2016; Sukhbaatar et al., 2015) are some of the QA varieties that benefited from DNNs. Many of the QA systems mentioned above like QA systems based on Semantic Parsing (Clarke et al., 2010; Kwiatkowski et al., 2010), IR-based systems (Yao and Durme, 2014), cloze-type (Kadlec et al., 2016; Hermann et al., 2015), factoid (Bordes et al., 2015; Aghaebrahimian and Jurčíček, 2016b) and non-factoid systems (Rajpurkar et al., 2016; Aghaebrahimian, 2018a) also have been improved by using DNNs.

There are at least three reasons why the use of DNNs attracted so much attention these days. First, a long-term goal of QA systems is to build general dialogue systems (Weston et al., 2016) and recently, end-to-end DNNs have shown excellent performance in dialogue systems. Second, the use of DNNs in QA helps to get rid of many cumbersome intermediate processes such as feature engineering at least in end-to-end systems. Finally, DNNs reduce the need for domain-specific knowledge and makes domain adaptation easier.

3.2 Question Answering: Architectures

A typical QA system includes some core and some peripheral subsystems. There are three core subsystems which can be almost always recognized both in sequential pipelines (Turmo et al., 2009) and end-to-end architectures. They are Question Processing, Passage Retrieval, and Answer Ranker components. Some QA systems may use other peripheral components or may join these components into a single one. In the following sections, we explain the core and some peripheral components of a typical QA system. Figure 3.1 schematically illustrates some of these subsystems.

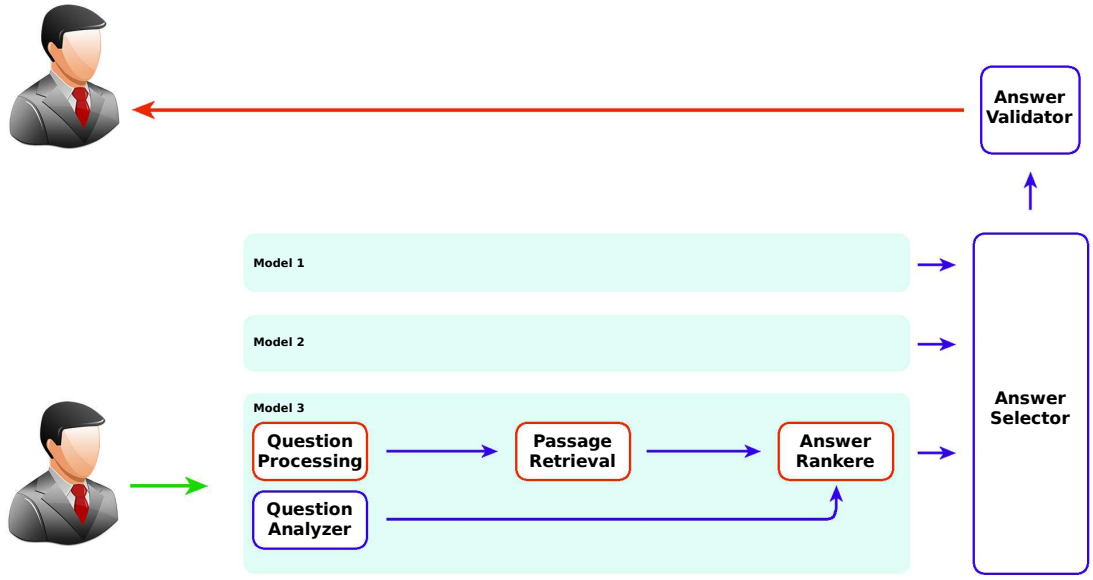


Figure 3.1: The core and peripheral subsystems in a typical QA system. The subsystems in red are core, and those in blue are peripheral.

3.2.1 Question Processing

Each question entails an explicit or implicit intention and contains possibly one or more entities. Recognizing these intentions and entities (i.e., question understanding) is the job of the Question Processing component.

In factoid QA systems, this component recognizes one property and one entity. In non-factoid QA systems, it extracts the intentions and their associated phrases and entities. For instance, in the question 'what decision did NFL owners make on May 21, 2013?', a question processing component realizes 'decision_made' as the intention and "NFL owners" and "May 21, 2013" as the entity and the time phrase which are associated to this intention.

In the SP approach, the Question Processing component treats each question as a combination of arguments and predicates (Berant et al., 2013). These combinations are trained using a machine learning technique to assign the highest rank to the combination which returns the correct answer.

In IR/IE approach, each question is processed either as a bag of words or as a low dimensional vector which is used to train a classifier to estimate a distribution over some predefined intentions given each question (Aghaebrahimian and Jurčíček, 2016a,b).

3.2.2 Passage Retrieval

Using the extracted keywords, syntactic structures (e.g., dependency or constituency trees) or statistical models from a Question Processing component, the Passage Retrieval component obtains a document or a short passage like a paragraph which hopefully contains the answer. These passages can be obtained from a specific repository like Wikipedia articles or directly from the Internet. Passage retrieval is usually done using a local IR engine or a commercial search engine.

Although an accurate Passage Retrieval component reduces the computational complexity and answer space for the next component (i.e., the Answer Ranker), sometimes finding a passage which contains the desired answer is as hard as finding the answer itself. So, some systems eliminate the need for Passage Ranker component from the QA pipeline by providing the correct passages (Rajpurkar et al., 2016).

3.2.3 Answer Ranker

The Answer Ranker component ranks the answers based on their relevance, accuracy, etc. and returns the highest or n-highest ones.

In factoid QA systems, the candidate answers are usually a named entity, a noun or a verb (Sun et al., 2005). In non-factoid QA, the constituents (e.g., noun phrases, adjective phrases, etc.) make the majority of the answers (Aghaebrahimian, 2018a, 2017c). The rest of the answers are non-constituents or any arbitrary string of consecutive tokens, and it makes the task of answer ranking much more difficult.

Symbolic and statistical techniques are the two broad approaches for Answer Ranking modules. A symbolic (or linguistic) approach assumes having access to rules, patterns or other possible linguistic measures like the similarity to determine whether a token or a series of tokens is the answer to a question. Similarity can be defined in terms of lexical (e.g., edit distance) (Aghaebrahimian and Jurčíček, 2016a) or syntactic measures (Moschitti and Quarteroni, 2010).

A statistical approach, in contrast, does not assume having access to such patterns. Instead, it has access to large quantities of data out of which it tries to derive some useful patterns. A conventional approach in statistical approaches is to check how probable is for an answer phrase to come with a question (Dumais

et al., 2002; Lin and Katz, 2003).

Question processing and answer ranking can be both benefited through typing information. Typing systems may use named entity types (Wu et al., 2005), typing system of an ontology (Hovy et al., 2001), or available types in a knowledge graph (Aghaebrahimian and Jurčíček, 2016a,b). Type enforcement is done through hand-crafted lexical rules (Prager et al., 2000), syntactic rules (Magnini et al., 2002) or machine learning classifiers (Punyakanok et al., 2004).

The Question Processing, the Passage Processing, and the Answer Ranker are almost always detectable in all QA systems. They constitute the essential components of a QA system. In addition to them, it is possible to see QA systems with some extended capabilities like the ones which follow:

3.2.4 Question Analyzer

The Question Analyzer component comes before the Question Processing component. It extracts temporal and spatial information and binds them to the answers in the list of its Answer Ranker component for retrieving finer answers (Kalyanpur et al., 2011; Hartrumpf et al., 2009). For instance, in the question 'what decision did NFL make on May 21, 2013', the Answer Analyzer extracts 'May 21, 2013' and helps the Answer Ranker component to increase the score of the candidate answers which are relevant to this date.

3.2.5 Answer Selector

Each QA system is only able to answer a specific type of questions. A system which performs superbly for IE-type questions performs treble for reasoning-type questions because the answers to reasoning-type questions are not explicitly mentioned in a corpus; an assumption which only holds true for IE-type questions. Therefore, ensemble models (Aghaebrahimian, 2017c; Ko et al., 2007; Mendes and Coheur, 2011; Clark et al., 2016) which include a stack of QA systems with different architectures often outperform each of the included systems individually. In these systems, each model generates an answer, and the final answer is chosen by an Answer Selector module which may use different possible approaches like ordering by likelihood, classification, etc.

3.2.6 Answer Validator

Providing answers with high precision is a critical task in some QA systems (Khani et al., 2016). In these systems, the answer validation component plays a crucial role. It is possible to validate answers using statistical or linguistic measures. In entailment checking as a statistical measure, an answer is validated by comparing it with its question to see if the former entails the latter (Wang and Neumann, 2007). Linguistic measures do the same job by reasoning over sentences which are generated out of the question and its answer using syntax rewriting rules.

4. Existing Datasets for Question Answering

In this chapter, we describe the datasets that we used for various parts of this work. We can roughly recognize all QA datasets as either sentence-level or word-level datasets. Given each question, a sentence-level QA dataset provides one or more correct sentences (Yao et al., 2013; Yang et al., 2015) while a word-level dataset offers one answer in the form of a single word (Richardson, 2013) or a span of consecutive words (Rajpurkar et al., 2016).

4.1 WikiQA

WikiQA (Yang et al., 2015) is a widely studied dataset for sentence-level QA. It is compiled from the query logs of the Bing search engine. The Wikipedia page selected for each query is used as the passage for that question. All sentences in the summary paragraph of the selected passage are used as the candidate sentences which in turn are presented to crowd workers for sentence selection. The questions in the dataset are mainly in six classes such as Location, Human, etc. The distribution of the classes and some statistics of the dataset are reported in Table 4.1 and Table 4.2 respectively.

Classes	Percentage of the questions
Location	12%
Human	16%
Numeric	22%
Abbreviation	1%
Entity	14%
Description	26%

Table 4.1: WikiQA classes and their proportions

4.2 TrecQA

TrecQA (Voorhees and Tice, 2000) is a standard and well-studied benchmark for answer sentence selection experiments. It is compiled using the data in TREC 8-13 QA tracks. Like WikiQA, the source of questions in TrecQA is users' log

	Train Set	Dev. Set	Test Set	Total
Questions	2,118	296	633	3,047
Sentences	20,360	2,733	6,165	29,258
Answers	1,040	140	293	1,473
Questions with no answer	1,245	170	390	1,805
Questions average length	7.16	7.23	7.26	7.18
Sentences average length	25.29	24.59	24.95	25.15

Table 4.2: WikiQA statistics

files. In both WikiQA and TrecQA, each question is mapped to more than one correct and several wrong sentences and QA systems are expected to return an ordered list of correct sentences.

The mapping between the questions to paragraphs for some of the questions in both WikiQA and TrecQA is not accurate due to indeterministic retrieval methods used for data retrieval in the first place. This is the reason why some of the questions in both datasets have no answer.

There is a modified version of the TrecQA dataset available in which the unanswered questions and questions with only one positive and negative sentences are removed from the development, and the test set divisions. The training questions in both the original and the modified versions are the same.

4.3 SimpleQuestions

SimpleQuestions (SQ) (Bordes et al., 2015) is another dataset for single-word QA. The answers in SQ are entity answers extracted from the assertions in a subgraph of Freebase (Bollacker et al., 2007) which is limited to 2 million entities (i.e., FB2M). Therefore, all answers to the questions can be found in FB2M[1]. The SQ dataset is a collection of 108,442 questions composed in natural language. Each question in the dataset is mapped to a triple of subject-predicate-object (a.k.a assertion) in the Freebase knowledge graph.

Crowd workers synthesize the questions in this dataset. They are posed to Freebase facts and are asked to synthesize a question. The SQ is a dataset for entity selection where the entities are limited to the entities in the system's knowledge graph. The dataset is randomly shuffled and divided into train set (70%), development set (10%) and test set (20%).

[1]For more information about FB2M and FB5M which are subgraphs of the Freebase please refer to Bordes et al. (2015)

Classes	Percent of the questions
Date	8.9%
Other Numeric	10.9%
Person	12.9%
Location	4.4%
Other Entity	15.3%
Common Noun Phrase	31.8%
Adjective Phrase	3.9 %
Verb Phrase	5.5%
Clause	3.7%
Other	2.7%

Table 4.3: The SQuAD classes and their proportions

4.4 SQuAD

Although some answers in SQ may contain more than one words, since the whole response is considered as one unit, SQ is regarded as a single-word dataset. In contrast, in the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) all answers are treated as chunks of words. The SQuAD is a dataset for QA in the context of machine comprehension. It includes 107,785 question-answer pairs posed by crowd workers on 536 Wikipedia articles. The answers in the SQuAD can be any span of consecutive words in a paragraph which comes with each question. The dataset is randomly shuffled and divided into train set (80%), development set (10%) and test set (10%). The answers in the dataset cover a wide range of classes (Table 4.3).

4.5 MS MARCO

MS MARCO, (Microsoft MAchine Reading COmprehension) is a large-scale dataset for reading comprehension and Question Answering. It contains 1,010,916 queries which are sampled from anonymized user queries. For each query ten context passages are extracted from web documents using the Bing search engine. Crowd workers generated 182,669 answers out of these passages only if a passage summary could answer the corresponding query. BLEU (Papineni et al., 2002) and ROUGE-L (Lin, 2004) are used to measure the performance of QA systems testes on this dataset.

4.6 Q2AD

The last dataset we used in our experiments is the Quora Question Answering Dataset (Q2AD) (Aghaebrahimian, 2017b) which is one of the contributions of this work. The Q2AD is a dataset for sentence- and word-level QA. It is composed of the questions which are posted on Quora Question Answering site. Currently, it is a small size dataset, and we intend to expand it using a crowdsourcing platform in the near future.

One of the unique characteristics of the Q2AD is the presence of multi-part answers. The answers in the Q2AD contain several sections and each section either as a sentence or as a span of words is located in different part of the accompanying passage. A full answer to a question in the Q2AD is the one which contains all the separate sections.

The other characteristic of the dataset is the authenticity and originality of the questions. While in many other datasets the questions are synthetic, both questions and answers in the Q2AD are generated by humans for an authentic and original purpose.

The process of dataset compilation for the Q2AD is described in Chapter 5. We summarized some statistics of all the datasets in Table 4.4 for more straightforward comparison.

	Train Set	Dev. Set	Test Set	type	evaluation
Original TrecQA	1229	82	100	sentence-level	MRR/MAP
Modified TrecQA	1229	65	68	sentence-level	MRR/MAP
WIKIQA	873	126	243	sentence-level	MRR/MAP
Simple Questions	75910	10845	21687	entity-level	Path Acc.
SQuAD	86228	10778	10778	word-level	F1/Exact match
MS MARCO	80000	10000	10000	Human generated	BLEU/ ROUGE-L
Quora	210	30	60	word-level sentence-level	MRR/MAP

Table 4.4: Comparative statistics of all the datasets

5. Quora Question Answering Dataset

In this chapter, we introduce Quora Question Answering Dataset (Q2AD), a small but progressing sentence-level and word-level QA dataset with authentic questions and multiple-part answers, which has been developed during the Ph.D. studies of the thesis author. He authored the dataset with the assistance of a few anonymous annotators. The content of this chapter is based on a paper (Aghaebrahimian, 2017b) presented and published by him at the TSD 2017 conference.

5.1 Introduction

In an answer sentence selection system each question is accompanied by a passage, and the task is to return a sentence or an ordered list of sentences from the passage as the answer given a question. An answer sentence extraction system may be used as a stand-alone or as a subpart of a word-level QA system either to provide a proof for where the final answer is extracted from or to decrease the complexity of search space (Aghaebrahimian, 2018a).

To address the issues mentioned in the last chapter about QA datasets, we present Quora Question Answering Dataset (Q2AD) which is compiled from the questions in Quora. Quora is a question answering site where people ask their questions, and other people answer them according to their expertise or experience. The answers in Quora have a high degree of variance since each question is answered by a variety of people with different perspectives.

In contrast to previous datasets in which the questions are synthetic or answer-oriented (Rajpurkar et al., 2016; Bordes et al., 2015), the questions in the Q2AD are authentic. It means that real people asked these questions for obtaining factual information. Moreover, the answers are generated for real questions to address real problems which makes the dataset more reliable in real-life QA systems.

The Q2AD is the only dataset with multi-part answers, both in sentence-level and word-level formats. In contrast to other datasets where the answers are either one entity (Bordes et al., 2015) or a span of consecutive words from one part of the accompanying context (Rajpurkar et al., 2016), the answers in the

Q2AD are generally multiple-part answers each from different sentences of their accompanying passages.

To establish a baseline for the Q2AD, we tested it using our state-of-the-art answer sentence selection system (Aghaebrahimian, 2017a) which is described in Chapter 7. We also used human annotation to establish an upper bound for the dataset. Our results show a wide margin between machine and human performance on the dataset which demonstrates the difficulty of the task.

In the rest of this chapter, we explain the process of dataset compilation in Section 5.2 and evaluate it in Section 5.3 before we conclude in Section 5.4.

5.2 Dataset Compilation

The Q2AD consists of 300 questions accompanied by their sentence-level and word-level answers. It is divided between the training set with 80% and test set with 20% of the questions[1]. Each question in the dataset is provided with a full-text context which contains the sentence-level and word-level answers to the question.

The sentence-level answers are complete sentences in full-text passages, and the word-level answers are any possible span of consecutive words in the sentence-level answers.

As shown in Table 5.1, the question *'How do I push myself to study in the afternoon?'* is answered by three full sentence-level and three word-level answers. The word-level answers are a span of consecutive words from the sentence-level answers. However, the choice of the boundary is entirely arbitrary, and each answer is from different parts of the context. As shown in the same table, there are no any common words between a question and its responses. This feature makes answer selection difficult for QA systems which work based on the overlap between queries and answers[2].

The Q2AD is compiled in three steps; Question Screening, Passage Selection, and Answer Annotation. These steps are explained in details in the following subsections.

[1] In our experiments we used the last 10% of data in training set for validation.
[2] For more samples from the dataset please see Appendix D.

Question	How do I push myself to study in the afternoon?
Full text passage	Many people have a down cycle after lunch. You can eat a light lunch which will help. That way your body isn't processing a heavy load of carbs and not as much energy is spent on the digestion of your meal.You can also build in a small amount of exercise: stretching, a short walk. Exercise helps digestion and helps improve your overall energy level.You can also deliberately set a timer for a work period. There is a method called the Pomodoro method you can check out which lets you set a clock and at the end of that time, you stop your work, rest for 5 minutes and then reset the clock. This method has been scientifically proven to help people be more productive.
Sentence-level answers	You can eat a light lunch which will help. You can also build in a small amount of exercise: stretching, a short walk. You can also deliberately set a timer for a work period.
Word-level answers	eat a light lunch exercise set a timer for a work period

Table 5.1: A sample question with its answers from the Q2AD. Correct sentences are full sentences from the context, and short answers are spans of words taken from correct sentences.

5.2.1 Question Screening

In the Question Screening step, a list of questions which are suitable for the dataset is compiled. Questions for inclusion in the dataset should be straightforward, well stated and eloquent. Besides, they should be answerable in at least one and at most three explicit sentences. To satisfy these conditions, to maintain a good variance in the dataset and to make sure that we choose our questions randomly enough, we underwent the following process.

At the time of dataset compilation, Quora had hosted around 200K answered questions. To collect the most popular questions, we did a weighted sampling to select 5000 questions. We defined the weights as the product of the 'Views' parameter and the number of answers associated with each question both normalized by the number of all questions.

We recruited a group of ten annotators for doing human annotation on the dataset. They were all native speakers of English and college graduate students. We asked the annotators to go through the questions one by one to eliminate questions which ask about more than one thing (e.g., which computer system should I buy? apple or Asus, why and preferably where?).

Afterward, they were asked to eliminate opinionated or subjective (e.g., what is the nastiest things happen to you recently?) and descriptive or procedural (e.g., how can I reinstall windows on my pc?) questions. The answer to these questions is usually very long. Besides, there is often, no way to answer them explicitly.

Finally, the annotators extracted questions which were self-explanatory and could be answered with no further clarification[3]. At this point we had around 4000 questions for processing in the next step, Passage Selection.

5.2.2 Passage Selection

In the last step, we compiled a list of questions. Our annotators chose a passage for each of these questions in this step. Passage selection is made on the merit of answering questions explicitly and providing enough context for answering them correctly.

A different number of people answer each question in Quora. Some of the questions are answered more than 100 times. It means that there are a large number of different answers with different length to each question.

A good passage for answering a question should be long enough to convey the message adequately. Too short passages do not provide enough context for answering questions. Hence, the first step in the passage selection is to eliminate too short passages. We rejected questions with passages less than 100 words in length. We did not care about too long passages since the system should be able to deal with it in real-life applications.

A good passage for answering a question should also be able to provide an explicit answer to their questions, and it can not be decided merely based on the 'Up-vote' statistic associated with them. The best full-text passage is not necessarily the one which has the highest up-votes. It is the one which contains correct answer sentences and short answers explicitly. Empirically, we proved this idea when we observed that more than 40% of the passages selected by our annotators were not from the answers with the highest 'Up-vote'. Therefore, given a question and its remaining full-text passages after the screening above, we asked our annotators to choose the best passage which satisfies the criteria mentioned above.

The passage of each question is extracted by at least two annotators to increase the confidence margin on the extracted passages. Since the annotators were trained with the same principles and all of them began from the highest up-voted answers and continued to the lowest ones, the chance of getting the same passage for each question from two or more annotators was high. We computed the

[3]Some users in Quora provides their questions with a comment which helps to clarify their question.

	train dataset	test data
Number of sentences	1212	302
Number of questions	240	60
One sentence	81	22
Two sentences	108	30
Three sentences	51	8
One answer	76	16
Two answers	104	33
Three answers	60	11
Average Sentence Length	60	11

Table 5.2: The Q2AD statistics.

Cohen's Kappa score for the inter-agreement among different annotators. The score was 86% which suggests high predictability and confidence in the process of passage selection. Since we needed only one passage per question, we extracted the question for which all annotators had chosen the same passage.

5.2.3 Answer Annotation

In the Answer Annotation step, sentence-level and word-level answers are annotated in the passages. The answers in the Q2AD are multifaceted. It means that a response to a question may contain different aspects which are expressed in multiple sentences. For this reason, the number of correct sentence-level and word-level answers are different for each question. The answers in the Q2AD may contain at least one and at most three different aspects (Table 5.2).

Given a list of questions accompanied by their passages, our annotators were asked to extract correct sentence-level and word-level answers. They were asked, first to choose the right sentences as sentence-level answers, and then to choose the right spans of words among the correct sentences as word-level answers.

As one could expect due to the fine passage selection in the last step, the inter-rater agreement (Kappa) in this step increased and reached 89%. To dissolve the disagreement among the annotators, we extracted the questions for which different annotators selected the same answers. Table 5.3 summarizes the length of questions, passages, and answers in the Q2AD.

Length	train dataset	test data
Questions: Max	31	30
Questions: Min	3	3
Questions: Mean	10.64	10.52
Full text: Max	337	302
Full text: Min	34	40
Full text: Mean	102.25	107.43
Sentence-level Answers: Max	51	59
Sentence-level Answers: Min	1	1
Sentence-level Answers: Mean	15.17	15.08
Word-level Answers: Max	33	17
Word-level Answers: Min	1	1
Word-level Answers: Mean	5.7	5.6

Table 5.3: The Q2AD questions and answer length

5.3 Evaluation

We used a state-of-the-art sentence-level QA system (Aghaebrahimian, 2017a) to measure the difficulty of the dataset and to establish a baseline. Given each question, the system generates an ordered list of word-level and sentence-level answers. Therefore, we used Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR) to measure the performance of the system.

To obtain the human performance on the Q2AD, we asked our annotators to select correct sentences for the questions in the Q2AD test set. Then, we computed MRR and MAP scores for each annotator, and we reported their average.

Table 5.4 summarizes the results of the Q2AD evaluation for random guess, upper bound and baseline experiments. In this work, we report the system performance only on sentence selection.

	MRR	MAP
Random guess	35.6	31.4
State-of-the-art	61.2	45.9
Upper bound	94.6	92.8

Table 5.4: Experimental results. State-of-the-art refers to (Aghaebrahimian, 2017a)

The wide margin between the random guess and the state-of-the-art results in Table 5.4 suggests the effectiveness of the approach used in (Aghaebrahimian,

2017a). However, there is a wide gap between machine and human performances which suggests there is still a lot of room for improvement.

An error analysis done on the responses shows that by increasing the number of answers in the answer set of each question the error rate decreases. It shows that most errors are attributed to single answer questions while least errors are attributed to triple answer ones.

5.4 Conclusions

In contrast to WikiQA or TrecQA, the connection between the questions and their passages in Q2AD is deterministic and direct. Answer up-voting by users strengthen this connection. So the answers in Q2AD are tailored toward the questions. Moreover and unlike to SQ or SQuAD, the questions in Q2AD are authentic and original because, in Quora, people ask questions to elicit useful information while in SQ or SQuAD crowd workers are asked to generate a question provided with an assertion or a paragraph.

This dataset poses a new challenge for QA systems for answering the questions which are seeking multi-part answers. Some questions require an answer which contains several sections each of which is taken from different parts of their accompanying paragraph. This feature to the best of the author's knowledge is absent in previous QA datasets.

All of these attributes make the Q2AD an ideal source for experimentation on open-domain Question Answering. In our future work, we intend to expand the number of questions in the Q2AD to an amount which makes more data-intensive approaches possible.

6. Structured Question Answering

Structure QA is the task of answering questions with the data provided in structured databases like relational databases or knowledge graphs. In this chapter, we describe our own original QA system based on the knowledge graph for answering simple open-domain questions. This chapter is taken from a paper (Aghaebrahimian and Jurčíček, 2016a) which was prepared by the thesis author together with his supervisor and was published at the Human-Computer Question Answering workshop at NAACL 2016.

6.1 Introduction

Simple open-domain QA is the task of answering questions using a knowledge graph (Bordes et al., 2015). Simple in this context does not mean that this is an easy task. It says that the answers can be obtained only by knowing one entity and one property[1]. Flexible and unbound number of entities and their properties in open-domain questions is an intimidating challenge for entity recognition. The Freebase (Bollacker et al., 2007), the knowledge graph which we used in our experiments, contains about 58 million entities and more than 14 thousands properties.

In our approach, we use a classifier for detecting properties and some heuristics based on the structure of the Freebase for recognizing entities in questions. The features of the classifier are defined using the words in the questions[2]. Of course, the most straightforward approach for doing so is to represent the words as one-hot vectors and train a classifier on them. However, representing words as discreet entities leads the classifier to disregard possible similarities between two tokens. This issue is known as the sparsity problem, and the low-dimensional vector representation of words known as embeddings partially solves this issue.

Word vectors or embeddings capture useful information about words and their relations to one another. Some studies show that there are meaningful connections among learned word vectors like gender (king → queen) or even major city of

[1]Entity is a thing, place or people and property is an attribute which is asked about a specific entity. For instance, in the question, *'What is the time zone in Dublin?'*, Dublin is an entity and time zone is a property.

[2]For a description of features in this system please refer to Appendix A

countries (Germany → Berlin) (Collobert et al., 2011). This attribute makes embeddings useful features for different NLP tasks especially those which require making decisions on the semantic contents of the texts. We use word embeddings as features in our Logistics Regression model. In our Neural models, we use two alternatives; learning the word vectors through training and using pre-trained word vectors.

We use the Freebase to recognize entities at test time. Defining a model for entity disambiguation on a single question instead of a whole dataset lets us scale the system up to a large knowledge graph irrespective to its size. We elaborate on entity recognition in Sections 6.3.2 and 6.3.3.

A study done by Bordes et al. (2015) shows that 86% of the questions in the WebQuestions dataset (Berant et al., 2013) are simple questions. WebQuestions is compiled using the Google Suggest API. It shows that a large number of questions asked on the Internet by ordinary people are simple questions, and it emphasizes the importance of simple QA systems. The best result of this task is 63.9% (Bordes et al., 2015) which suggests simple QA is still an unresolved task in NLP.

6.2 Method

We define P and E as the space of all properties and entities in a question. For each question like *'What is the time zone in Dublin?'*, we intend to find the tuple $(p, e) \in P \times E$ for which p's probability and e's score with respect to some features are maximal. We would like to get *'/location/location/time_zones'*[3] as the best property and *'/en/Dublin'* as the best-matching entity in this question.

We decompose the learning model into two steps, namely; property detection and entity recognition. In property detection, we decide which property best describes the purpose of a given question. In this step, we model the assignment of properties given questions using the probability distribution in Equation 6.1. We use Logistic Regression technique to train the model and use the model for assigning an n-best property list to each question at test time.

$$P(p|q) = \frac{\exp(\omega_p^T \phi(q))}{\Sigma_{p_i} \exp(\omega_{p_i}^T \phi(q))} \tag{6.1}$$

[3]i.e., /Type 1/Type 2/Predicate

Given a question q , the aim is to find an n-best property list which best describes the content of q and generates the correct answer when querying against the knowledge graph. ϕ in Equation 6.1 is a feature set representing the questions in vector space, and ω is the parameters of the model. p is the set of properties in SimpleQuestions(SQ) (Section 4.3) which we used to evaluate our system. There are 1629 properties in the training set of SQ. In Figure 6.1 the distribution and some of the most common properties of the Freebase as the source of SQ is illustrated.

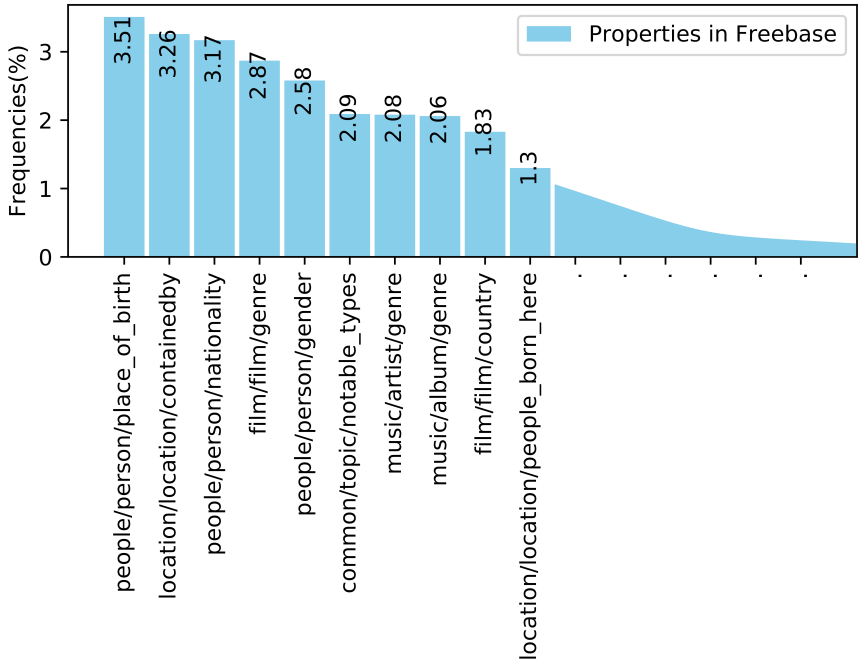


Figure 6.1: The long tail of properties in the Freebase

In the second step, i.e., entity recognition, we detect and disambiguate the primary entity of a question. We use a set of heuristics for assigning the best-matching entity to each test question at test time. Entity recognition consists of entity detection and entity disambiguation.

In entity detection, we try to detect the primary entity of a question. A typical question usually contains tokens that all are available in the Freebase while only one of them has the primary focus. For instance, in the question *'what is the time zone in Dublin?'*, there are eleven entities all of which are available in the Freebase (*'time, zone, time zone, ..., Dublin'*) while the focus of the question is on *'Dublin'*. (Please see Figure 6.2)

Figure 6.2: The spans of different available entities in a question

The detected entities are mostly ambiguous. Given an entity like *'Dublin'*, we want to know which Dublin (i.e., Dublin in Ireland, Dublin in Ohio, etc.) is meant in the question and it is not a trivial task. An increase in the number of tokens in questions leads to an exponential rise in the number of possible entities among which we should disambiguate. In Figure 6.3 the growth of entities by an increase in the length of questions is depicted. To help the system with entity disambiguation, we use some heuristics to increase the chance of correct entities.

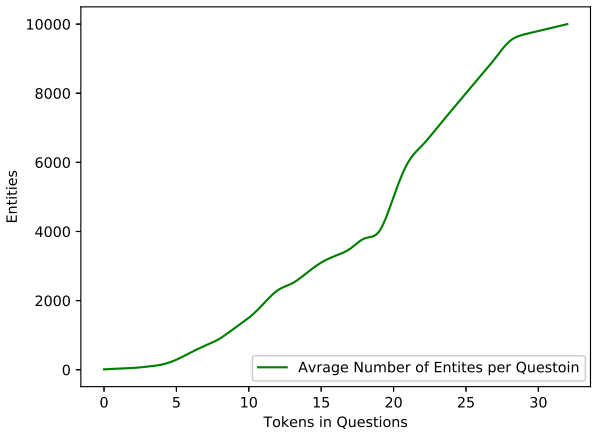


Figure 6.3: The graph of the exponential growth of ambiguous entities by an increase in the tokens of a question. The statistics are extracted from the questions in the SQ dataset.

Having an n-best list of properties assigned to each question, we use two heuristics namely type and similarity heuristics to select the best matching entity.

- The type heuristic enforces the type of answers obtained by querying (p_q, e_p) to be equal to the expected type of p_q. p_q is the detected property for question q and e_p is the matched entity for that property.
- The similarity heuristic dictates that e_p score which is the lexical similarity ratio (i.e. edit distance) between the string values of *'name'* and *'id'* (see Figure 2.5) properties connected to it should be maximal among all other e_ps .

The type heuristic helps in detecting the primary entity of a given question among other entities. Despite the assigned properties, each question has E number of valid entities. By valid we mean entities which are available in the Freebase.

After property detection, a set of n-best properties is assigned to each question each of which has no or at most one *'expected_type'*. The Cartesian product between the n-best properties and the E valid entities gives us $N \times E$ tuples of *(property, entity)*. We query each tuple against the Freebase and obtain the respective answer. Each of the answers has a set of *'type'*s. If the *'expected_type'* of the property of each tuple was available in the set of its answer's *'type'*s, the type heuristic for the tuple holds true otherwise false.

The similarity heuristic helps in entity disambiguation. Each entity has an *'id'* and a *'name'* property. Ambiguous entities usually have the same *'name'* but different *'id'*s. For instance, entities *'/m/02cft'* and *'/m/013jm1'* both have *'Dublin'* as their *'name'* while the *'id'* for the first is *'/en/dublin'* and for the last is *'/en/dublin_ohio'*. This is also the case for more than forty other different entities whose *'name'* are *'Dublin'*. In this case, the similarity heuristic for entity *'/m/02cft'* holds true because among all other entities, it has the minimal edit distance (i.e maximal similarity) ratio between its *'name'* and *'id'* values. It is possible that the content of *'id'* property for an entity is the same as its *mid*. In such cases, instead *'id'*, we use *'alias'* property which contains a set of aliases for entities.

6.3 Training

At training time, we have training questions accompanied by their Freebase assertions each of which includes an entity, a property, and an answer. Entities and answers are in their mid formats. We also have access to the Freebase (data dumps, 2015) through an MQL query API.

First, all questions are fixated on 20 tokens. Then we chunk them into their symbols and compute $\phi(q)$ by replacing each token with its vector representation. To train our classifier, we assign a unique index to each property in the training data set and use them as a label for the training questions. Given a test question at test time, we first get the n-best properties using our trained model as it is explained below.

6.3.1 Property Detection

In property detection, we train a model which assigns an n-best property list to each question based on its content. We use three models for this purpose namely Logistic Regression[4], a simple Neural Network[5], and a Convolution Neural Network[6]. The task in all of these models is to assign a property to each question.

In the Logistic Regression model, the features are the words in questions replaced by their word vectors taken from pre-trained Word2Vec vectors (Mikolov et al., 2013). Although it is not an efficient representation of questions, this model obtains a competitive result in the SQ dataset. Figure 6.4 illustrates the Logistic Regression model.

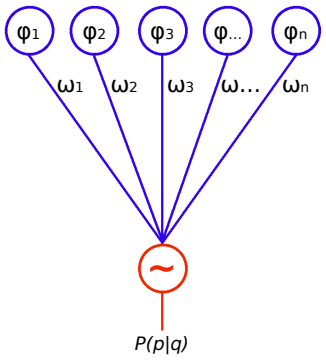


Figure 6.4: The Logistic Regression model. Each circle represents a token, ϕ is the features, and ω is the weights. We use the Sigmoid function as the non-linearity ($\sim$) in the model.

In the next experiment, we added more layers to our Logistic Regression model to enhance the performance of the classifier. Figure 6.5 illustrates the Neural Network model.

[4]For a detailed description of this model please see Section 2.1.1
[5]For a detailed description of this model please see Section 2.1.2
[6]For a detailed description of this model please see Section 2.1.3

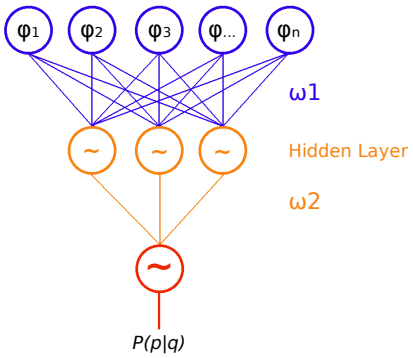


Figure 6.5: The Neural Network model. The non-linearity in the hidden layer is sigmoid and in the last layer is softmax.

Finally, to enhance the model even further, we use a Convolution Neural Network (CNN) with different filter sizes and with a max pool layer on the top. The architecture of our CNN model is similar to Yoon Kim (Kim, 2014) with some minor modifications.

The CNN model contains four consecutive layers. The first layer embeds words into a low dimensional vector representation. In this layer, we adopt two approaches to learning embeddings. In the first approach, we let the graph to learn word embeddings directly from data by initializing it with a random uniform distribution. In the second approach, we use pre-trained Word2Vec word embeddings. In both methods, we keet the model updating the embeddings through training. The two alternatives show no significant difference in the final results.

The second layer slides a convolution window with different sizes over the embeddings and the third layer max pools the result into a vector which is fed into a softmax layer for classification in the last layer. For convolution layer, we use one, two and three-size windows and for the next layer, we try average and max pooling.

Finally, we use a fully connected softmax layer at the end. We train a model in training time, and then we use it to obtain an n-best property list at test time. Figure 6.7 illustrates the learning curve of the three models mentioned above on the SQ validation set. Figure 6.6 illustrates the CNN model.

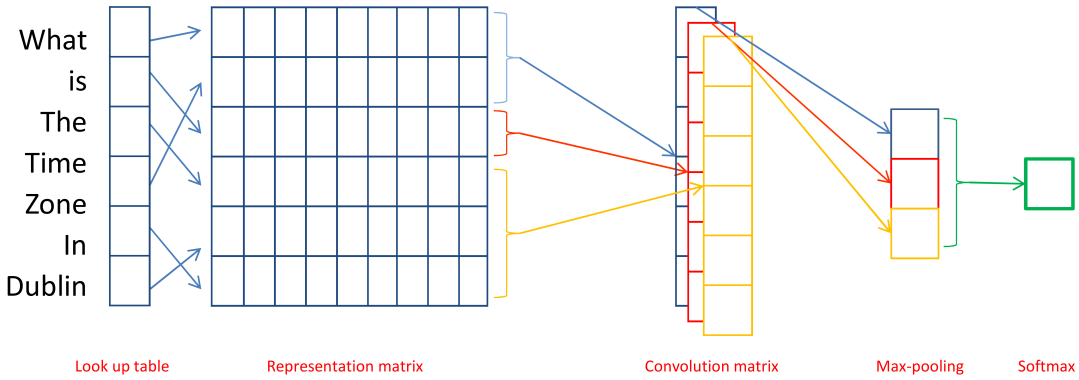


Figure 6.6: The CNN model

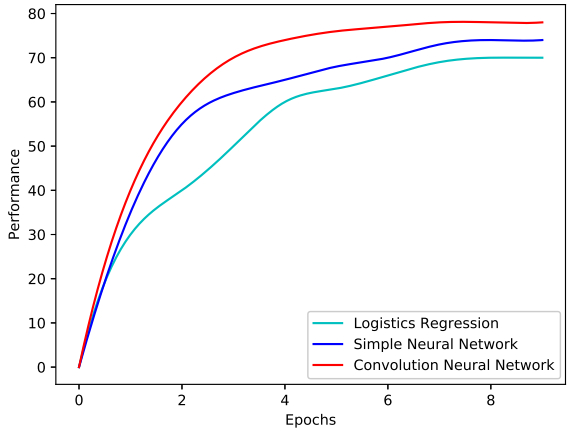


Figure 6.7: The leaning curve of the Logistic Regression, Neural Network, and Convolution Neural Network models as the function of the number of training epochs

Having the properties, the next step in the model is entity recognition which includes entity detection and entity disambiguation.

6.3.2 Entity Detection

At the test time, instead of relying on external lexicons for mapping surface forms to logical forms, we match surface forms and their mids directly using the Freebase. For entity detection, we extract the spans in questions which are available in the Freebase. To do so, we slide a flexible size window over each question and obtain all possible spans with all possible sizes. We query the live and full version of the Freebase using Meta-Web Query Language (MQL).

We query the entity mid of each span against the Freebase using the following query.

```
{"mid":[],
        "name": a span of word(s)}
```

This query returns the mid-list of all entities which have the same name as the provided span. We have two alternatives to obtain entity mids, namely greedy and full.

In the greedy approach, only the longest valid entities are reserved, and the rest which may be still valid is disregard. In full approach, however, all the entities are reserved. For instance, in a simple span like 'time zone', while greedy approach returns only *'time zone'*, full approach returns *'time'*,*'zone'* and *'time zone'*.

The spans with at least one entity mid are recognized as valid entities. For each mid, we again query the Freebase to get the id and the types of the entity using the following query.

```
{"mid":mid,
        "type":[],
        "id":[]}
```

For each mid, this query returns a list of ids and the associated types. As described in Section 6.2, we generate the Cartesian product between the n-best properties which is generated using the models in the previous section and the mids of the current question. Enforcing the type heuristic on these products distinguishes relevant entities from irrelevant ones. In other words, we remove the products in which the expected type of the property is not available in the list of the types of answer entities.

6.3.3 Entity Disambiguation

Detected entities in the last step in many cases are ambiguous. Entities in large knowledge graphs each have different meanings and interpretations. In a large knowledge graph, it is possible to find *'Dublin'* as the name of a city as well as the name of a book. Moreover, when it is the name of a city, that name is not still unique as we saw in the earlier section. We use lexical similarity between *'id'* and *'name'* properties connected to that entity as a heuristic to get the best-matching entity. Therefore for all remained (p_q, e_p) combinations from the previous step we compute the edit distance between the name and id associated to the mid and take the combination with the smallest distance.

6.4 Experiment

We use the following algorithm to train and test our system.

Algorithm: Property-driven QA

Input: train, validation and test sets, KG
Output: assertions for test questions

Training:
Train a classifier on train and validation sets

Testing:
For each question in the test data set

- Get n-best properties using the trained model
- Query KG to get the entity mids of word spans in the question
- Entity recognition (type and similarity heuristics)
- Query KG to get the answer

return assertion(entity, property, answer)

We tested our system on the SimpleQuestions (SQ) data set (Bordes et al., 2015)[7]. To make our results comparable to the results of SimpleQuestion authors, we conducted our experiments on the official separations of the dataset. The input for the training step in our approach is the training and validation sets with their knowledge graph assertions. Using Word2Vec toolkit (Mikolov et al., 2011), we replaced the tokens in the datasets with their vector representations and used them as $\phi(q)$ in our model. We pruned questions with more than twenty tokens in length and fixate shorter ones by adding extra <.> token. We already did some simple pre-processing jobs on the input data such as removing non-alphanumeric characters.

[7]For more about this dataset please see 4.3

Using these features and the model described above, we trained a classifier using Logistic Regression, Neural Network, and CNN technique. We used the trained classifier at test time for detecting 100-best properties for each test question. For Neural Network, we used two hidden layers each with 1024 neurons, and for CNN we used the same number of neurons for convolution and softmax layers.

We tried to enforce regularization on weight vectors, however as already tested in (Zhang and Wallace, 2015) it did not affect the final results in a meaningful way. We also included a new channel in our CNN using POS tags of tokens, and it improved the last model but not significantly. A comparative illustration of the three models is depicted in Figure 6.7 and the results of hyper-parameter tunings are reported in Appendix A.

The trained classifier, test questions and the Freebase knowledge graph (February 2016) are the inputs at test time. Performing entity detection and disambiguation on the spans of a question using the detected properties, we obtain a path (i.e., an entity and a property) using which we can evaluate the system performance.

We used the Path Accuracy for evaluating the system. This is the same evaluation metric which is used by the dataset authors. We obtained our best validation accuracy using the greedy approach for entity recognition and 128-dimensional embeddings for property detection. Using the same configuration, we reported the accuracy of our system on test data.

6.5 Results

For training our system, we only used SimpleQuestions. Then, we reported the results of official test data separation. The results of our system are reported in Table 6.1.

We reported the accuracy of property detection and overall system separately. In these series of experiments, the improvement in overall system accuracy is due only to the gain on property detection. Although our system makes a query on the whole knowledge graph, to make sure our results are comparable, we eliminate entities which are not available in FB5M. In this settings, with 99% coverage, we obtained 61.2% accuracy in our Logistic Regression model which is competitive to the results in (Bordes et al., 2015) when training on the same dataset (61.6%). Our Neural Network system obtained 63.89% accuracy which

	trained on	property Acc.	over all Acc.	knowledge graph
Bordes et al.	SQ	-	61.6	FB5M
Constraint-based(LR)	SQ	69.80	61.20	Full FB
Constraint-based(NN)	SQ	74.08	**63.89**	Full FB
Constraint-based(CNN-1)	SQ	78.02	**65.16**	Full FB
Constraint-based(CNN-2)	SQ	78.82	**65.19**	Full FB

Table 6.1: Experimental results on the test set of the SimpleQuestions (SQ) dataset. LR stands for Logistic Regression, NN for Neural Network, CNN-1 for Convolution Neural Network with one channel and CNN-2 for the CNN with two channels.

is the same with (Bordes et al., 2015) best results when they trained on three training data sets (WebQuestions and paraphrase data sets in addition to Simple Question). Finally, our CNN model obtained 65.19% accuracy only trained on SimpleQuestions. Since we work on the full knowledge graph, we hope that our system can answer every possible simple question posed on the full graph of the Freebase.

6.6 Error Analysis

Although the Logistic Regression and Neural Network models converged after 80% of the training data, as the learning curve in Figure 6.8 suggests, the CNN model has still more capacity for learning. Therefore providing more training data could improve the performance of this model.

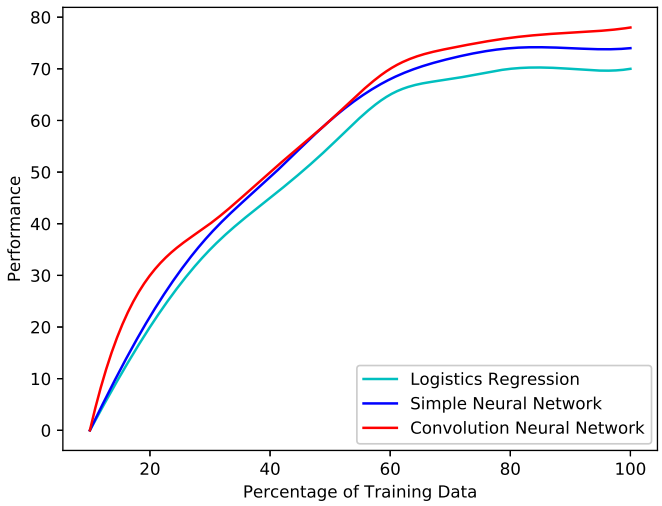


Figure 6.8: The learning curve of Logistic Regression, Neural Network and Convolution Neural Network models as the function of the number of training data

The errors made by the system can be categorized into three major categories: Errors made on property classification, errors due to entity detection, and errors due to entity disambiguation.

Around 20% of the errors are made directly by the property classification. The mistake made in the classifier is mainly due to either too identical properties or lack of enough training data for less represented properties. About 8% of the errors in this part is due to wrongly assigned very similar properties like 'music/artist/genre' and 'music/album/genre' which is a characteristic of the dataset. Other 5% of the errors are made because of the lack of enough training data on rarely-used properties such as 'geography/glacier/terminus' or 'socialdance/-dancer/styles.' These properties are in the long tail of the properties and are not used very often. These errors can be eliminated by including more training data for these under-represented properties.

The reaming 7 to 8% errors made by the classifier are attributed to imperfectly trained word vectors and the lack of enough training data especially for the CNN model (See Figure 6.8).

The entity detection is the source of the second class of errors due to the typing system of the Freebase. In some cases, the assignment of the expected type to properties seems non-deterministic like the assignment of 'genre' and 'book' both to $book/book - edition/publisher$. Nevertheless, the proportion of the errors made by this module is not noticeable.

Ambiguous entities make the third class of errors. In this errors the detected property and the surface name of the entity are correct, but the recognized mid is not. A more powerful entity linking approach can resolve these errors.

6.7 Conclusions

The class of entities in open-domain applications is open and expanding. Training a statistical model for classifying millions of entities is practically infeasible because it is costly and there is not enough training data available. Since entity decisions are made on one question and does not affect the next one, the entity model can be optimized for each question at test time and on the heuristics specific to that question.

In contrast, the class of properties is closed, and the property decisions usually are global ones. Therefore, a property model can be optimized once at training

time and for all the questions. In this way, by making decisions on properties first, we decrease the decision space for entities by a large extent, and it makes our approach insensitive to the size of the knowledge graph. As we demonstrated in our experiment, optimizing entity recognition model on a single question lets the system scale efficiently to large knowledge graphs.

Dependence on predefined lexicons limits the scope of language understanding only to those predefined ones. In our approach, we do not use any data set or lexicon for entity recognition. Instead, we disambiguate entities by querying the knowledge graph at test time. Then, we apply some heuristics to entities to get the correct entity for each question. In this way, we can expand the domain of language understanding irrespective to the size of the knowledge graph.

While our system is tested on the Freebase which contains more than 50 million entities, other systems limit their entity recognition on two subsets of this knowledge graph which contain only two and five million entities each. To reiterate how difficult is to scale up a system to such massive number of entities, we refer to an experiment done by Bordes et al. (2015) in which scaling the experiment from two million entities to five million entities deteriorates their system accuracy from 62.7% to 62.2%.

7. Sentence Selection

In this chapter, we introduce the Constrained Deep Neural Network (CDNN) as a simple deep neural model for sentence-level answer extraction. CDNN compounds neural reasoning with symbolic constraints to enhance final predictions. On a well-studied dataset for sentence-level answer selection, our model improves state of the art in answer sentence selection significantly. The content of this chapter is the shortened text of a paper (Aghaebrahimian, 2017a) which was prepared by the thesis author while he was working on his Ph.D. thesis.

7.1 Introduction

A typical Question Answering (QA) system consists of three primary components; Passage Retrieval, Sentence Selection, and Answer Extraction (Tellex et al., 2003). In this chapter, we focus on Sentence Selection.

The Sentence Selection component or a sentence-level QA system is designed to extract a subset of sentences in a passage given a question. If a more specific answer is expected, another component (i.e., Answer Extraction or word-level QA) extracts a word or a span of nearby words from the selected sentences.

The Sentence Selection component is beneficial in at least two ways. First, it helps the Answer Extraction component by eliminating non-relevant sentences and reducing the search space. Second, it provides a sentence which can be used as evidence for the final answer. Some other Natural Language Processing (NLP) tasks such as paraphrase detection (Yin et al., 2015b) benefit from this formulation of QA too.

Deep Neural Networks (DNN) has been shown to outperform traditional machine learning algorithms in many NLP tasks. A natural choice for sentence selection using DNNs is the Hinge approximation approach in which the weights associated to correct question-answer pairs are increased, and those associated to wrong pairs are decreased (Rao et al., 2016; Santos et al., 2014) through training. We extend this idea to integrate the number of shared patterns of question-answer pairs into the model.

Such a large DNN has millions of parameters that should be adequately trained, and It requires a large number of samples which are not available in regular datasets. However, DNNs work astonishingly well with transfer learn-

ing (i.e., training a model on a sizeable general-purpose dataset and optimize it using a smaller specialized dataset for a specific task). To provide the network with enough training samples, we use SQuAD (Rajpurkar et al., 2016) to train our model as a mean of transfer learning, and we show that the model performs remarkably better than previous best models.

7.2 Architecture

In a sentence selection experiment, we want to estimate a probability distribution over all sentences given each question and to get the sentence with the highest probability.

$$s_{best} = argmax_s \quad p(\mathbf{s}|\mathbf{q}) \tag{7.1}$$

To compute the probability, in the first step, we need to project the questions and sentences into n-dimensional space. To do this, we used the Recurrent Neural Network (RNN) (Elman, 1990) architectures to encode textual strings into vector representations. Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Cho et al., 2014) are two widely studied variants of RNNs. In our study, we used LSTM cells since they showed better and more stable performance in our experiments[1].

To provide LSTM layers with their inputs, in the first layer, we used a lookup table to cast words into word embeddings (Equation 7.2). $\mathbf{W}$ in Equations 7.2 is the one-hot representation of the words in sentences whose production with pre-trained embedding matrix $\mathbf{E}$ generates word vector $\mathbf{W}_{i,t}$ for the words in the i$^{\text{th}}$ sample sentence each in time step t.

In the next layer, a forward RNN layer accepts word vectors and generates a sequence of vectors for each time step. A similar RNN does the same job but in opposite order to generate backward RNN vectors. We did max pooling (**MP**) (Equations 7.4 and 7.6) over RNN vectors to get the most relevant features and then concatenated them in Equation 7.7. The final result of this process, $\mathbf{S}$ is a forward and backward vector representation of textual strings. We used this architecture to encode our questions.

[1]For a detailed description of this model, please see Section 2.1.4

$$\mathbf{W}_{i,t} = \mathbf{E}^{\intercal}\mathbf{W}_k \tag{7.2}$$
$$\overrightarrow{\mathbf{S}}_{i,t} = RNN(\overrightarrow{\mathbf{S}}_{i,t-1}, \mathbf{W}_{i,t}) \tag{7.3}$$
$$\overrightarrow{\mathbf{S}}_{i} = \mathbf{MP}(\overrightarrow{\mathbf{S}}_{i,t}) \tag{7.4}$$
$$\overleftarrow{\mathbf{S}}_{i,t} = RNN(\overleftarrow{\mathbf{S}}_{i,t+1}, \mathbf{W}_{i,t}) \tag{7.5}$$
$$\overleftarrow{\mathbf{S}}_{i} = \mathbf{MP}(\overleftarrow{\mathbf{S}}_{i,t}) \tag{7.6}$$
$$\mathbf{S}_i = [\overrightarrow{\mathbf{S}}_{i}; \overleftarrow{\mathbf{S}}_{i}] \tag{7.7}$$

Answer sentences are encoded like questions with an additional attention layer, which helps the encoder to recognize the most relevant features by emphasizing on critical points of the answer sentence given each question. Therefore, similar to the question encoder explained above, the sentence encoder receives the initial embeddings from a lookup table over a pre-trained embedding matrix. The forward RNN in the next layer transforms the embeddings into a sequence of vectors which finally are fed into an attention layer with attention on source sentence vectors (Equation 7.8). The resulted vector then is max pooled to be eliminated from non-relevant and useless features. The same process is done for the backward RNN, and the resulting vectors are concatenated.

$$\begin{aligned}
&\mathbf{W}_{i,t} = \mathbf{E}^{\intercal}\mathbf{W}_k \\
&\overrightarrow{\mathbf{T}}_{i,t} = RNN(\overrightarrow{\mathbf{T}}_{i,t-1}, \mathbf{W}_{i,t}) \\
&\overrightarrow{\mathbf{T}}_{i,t} = ATT(\overrightarrow{\mathbf{T}}_{i,t}, \mathbf{S}_i) \qquad (7.8) \\
&\overrightarrow{\mathbf{T}}_{i} = \mathbf{MP}(\overrightarrow{\mathbf{T}}_{i,t}) \\
&\overleftarrow{\mathbf{T}}_{i,t} = RNN(\overleftarrow{\mathbf{T}}_{i,t+1}, \mathbf{W}_{i,t}) \\
&\overrightarrow{\mathbf{T}}_{i,t} = ATT(\overleftarrow{\mathbf{T}}_{i,t}, \mathbf{S}_i) \qquad (7.9) \\
&\overleftarrow{\mathbf{T}}_{i} = \mathbf{MP}(\overleftarrow{\mathbf{T}}_{i,t}) \\
&\mathbf{T}_i = [\overrightarrow{\mathbf{T}}_{i}; \overleftarrow{\mathbf{T}}_{i}]
\end{aligned}$$

All sentences including correct and wrong sentences are encoded using this procedure. In the end, question vectors in addition to right and wrong answer vectors are ready to be used as the inputs of a Hinge objective function. Using this function, we try to maximize the similarity in right question/sentence sets while minimizing it in wrong question/sentence sets. Therefore, the next step is to measure the similarity between questions and sentences in the right and wrong sets.

The similarity between two vectors can be estimated via different approaches such as Jaccard, Cosine, Polynomial or Manhattan, etc. But it seems that an adequate similarity measure highly depends on the task. We experimented several similarity measures (please see 2.3) including Cosine, exponential, etc. and we adopted Geometric mean of Euclidean and Sigmoid Dot product (GESD) (Feng et al., 2015b) which seems to outperform other similarity measures in this model.

GESD linearly combines two other similarity measures called L2-norm and inner product. L2-norm is the forward-line semantic distance between two sentences, and inner product measures the angle between two sentence vectors (please see Equation 7.10)[2].

$$GESD(\mathbf{q}, \mathbf{s}) = \frac{1}{1 + \exp\left(-(\mathbf{q} \cdot \mathbf{s})\right)} * \frac{1}{1 + \|\mathbf{q} - \mathbf{s}\|} \tag{7.10}$$

To distinguish correct question/sentences from wrong ones, we need to train their vectors in a way which increases the similarity for right and decreases it for wrong question/sentences. Hinge objective function does the trick for us (Equation 7.11). We also modify it in a way that not only considers the similarity between questions and their sentences but also enforces their pattern similarity as a hard constraint (see Equation 7.11).

$$\ell = \sum_i \max(0, m + \beta * GESD(\mathbf{S}_i, \mathbf{T}_i^-) - \alpha * GESD(\mathbf{S}_i, \mathbf{T}_i^+)) \tag{7.11}$$

[2]For a comparative analysis of various similarity measures in this experiment, please refer to Appendix B

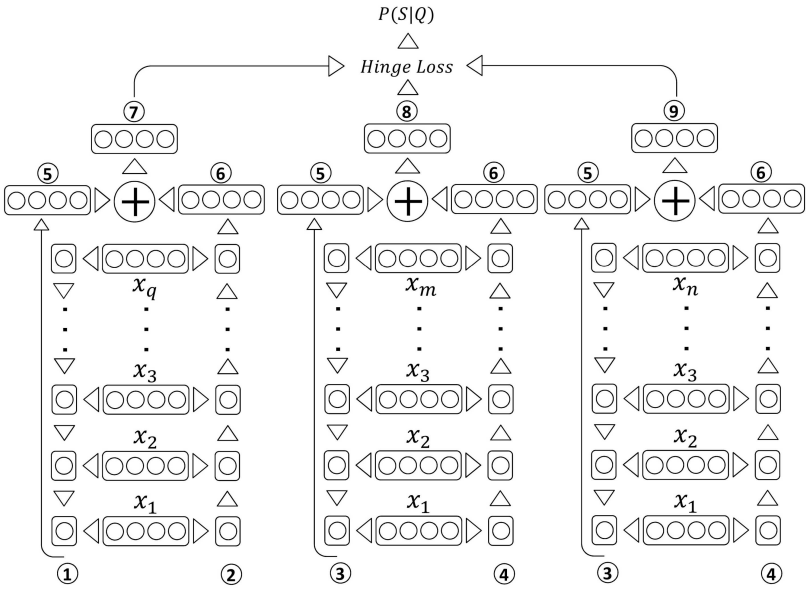


Figure 7.1: Abstract model architecture. Numbered components in the figure are 1-Forward LSTM, 2-Backward LSTM, 3-Forward Attentive LSTM, 4-Backward Attentive LSTM, 5-Backward max-pooling, 6-Forward max-pooling, 7-Question vector, 8-correct sample vector, 9-wrong sample vector.

Any similarity between a question and its correct sentence decreases the loss and vice versa (i.e., the loss increases with the similarity between a question and its wrong samples.). Parameter **m** which is a value in range 0.01 and 0.1 makes a trade-off between the number of mistakes in true and false classification in the model. $\mathbf{s}^+$ are correct, and $\mathbf{s}^-$ are wrong sentences. α and β are the numbers of shared patterns between a question and its right and wrong sentences respectively. Finally, $GESD$ (Equation 7.10) is the similarity function that computes the similarity between the vectors of questions and sentences. After training, the model estimates a probability for each pair of question and sentence (i.e., $p(question|sentence)$).

Figure 7.1 describes the equations above schematically. In training a sentence-level QA system, correct and wrong sentences are equally informative. Therefore to learn the probability associated with each question and sentence either right or wrong we should feed the network with right and wrong sentences at the same time to let the network to learn to contrast them.

Based on the equations above and as shown in Figure 7.1, questions are passed through four layers. The first layer is an embedding layer. The word vectors in this layer are initialized with pre-trained 300-dimensional Glove vectors (Pennington et al., 2014a).

In the second layer, two rows of LSTM cells are allocated to each question to form a forward and backward representations. Given a question, in the forward LSTM, each LSTM cell receives the output of its previous cell as the input layer. The process repeats from left to right to cover all the tokens. The first LSTM gets the embedding layer output as its input, and the output of the last LSTM is fed into the max-pooling layer.

In the backward LSTM, the same question is fed into the same LSTM, but in reverse order (i.e., from right to left). The outputs of the forward and backward LSTMs then go through a max-pooling layer. Similar to Santos et al. (2014), the max-pooling layer is a column-wise max operator that estimates the importance of each token given the surrounding context. Both forward and backward LSTM vectors are passed from the max-pooling layer, and the resulting vectors are concatenated.

To generate correct and wrong vectors right and wrong samples are passed through a similar network but with attention LSTM cells instead of regular ones over corresponding questions. In the end, we have a pair of correct and wrong sentence vectors for each question. By putting the similarity of right and wrong pairs into the loss function, we can train the network to maximize the similarity between questions and the corresponding correct sentences and to minimize the similarity between questions and wrong sentences.

Finally, we rearrange the scores generated by the model by a coefficient of question-sentence shared patterns similar to what we use in the loss function. In the end, we use a softmax to turn the generated logits into probabilities.

7.3 Dataset

We used two widely studied datasets for sentence selection, namely TrecQA[3] and SQuAD[4]. We used the TrecQA dataset to evaluate the performance of the system on sentence selection and the SQuAD to assess the system performance on transfer learning for sentence selection task. Although the SQuAD is not designed originally for answer sentence selection, the correct answer sentences can be easily extracted using the pointers defined on the beginning and ending characters of correct word-level answers.

Wrong sentences are collected using two different settings, which seem to cause

[3]For more about this dataset, please refer to Section 4.2
[4]For more about this dataset, please see Section 4.4

no real difference in the final results. In the first setting, given a question, we detected the correct sentence as explained above and used the other sentences in the same paragraph as wrong samples. In the other setting, we sampled wrong sentences for a given question from all the paragraphs disregarding to which paragraph the question belongs. In the end, we had one correct and up to five wrong samples for each question.

The original test set of the SQuAD is unseen and is not available online. Hence, for our experiments, we used the original training set for both training and development purposes. For testing purpose, we used the original development set. Since this is the first time that the SQuAD is being tested in a sentence selection experiment, we established a simple baseline for sentence selection in our test set. We counted the number of shared uni, bi, and trigrams in each question and sentence and assigned the sentence with the most shared patterns to the question. Then we reported the Accuracy of this method as the baseline.

7.4 Experimental Results

We used 128-dimensional LSTM cells and set the margin of the loss to 0.05. We used Adam (Kingma and Ba, 2014) to optimize the parameters using SGD. As the curve in Figure 7.2 shows the model benefits from all the training data.

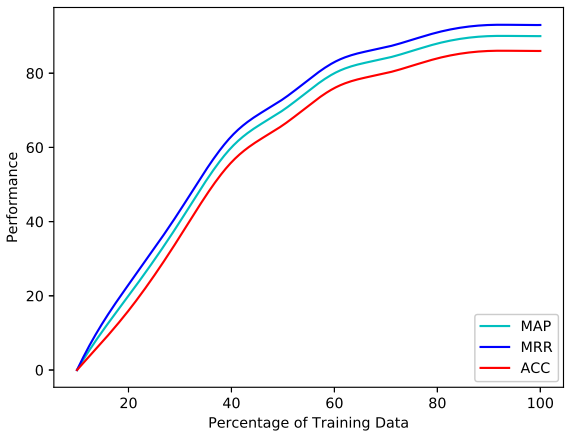


Figure 7.2: The learning Curve of the model as function of the percentage of training data

Given each question, there is more than one correct sentence in the TrecQA dataset. Therefore, the output of the model for this dataset is an ordered list of sentences which requires us to use the Mean Average Precision (MAP) and

	trained on	test set	MRR(%)	MAP(%)	ACC.(%)
state-of-the-art	TrecQA	TrecQA-mod	87.7	80.1	-
state-of-the-art	TrecQA	TrecQA-org	83.4	78.8	-
Current Paper	SQuAD + TrecQA	TrecQA-mod	86.2	73.2	-
Current Paper	SQuAD + TrecQA	TrecQA-org	**89.5**	**79.5**	-
Baseline	SQuAD	SQuAD	-	-	69
This work	SQuAD	SQuAD	93.4	90.1	**86**

Table 7.1: Experimental results. The TrecQA-org is the original TrecQA, the TrecQA-mod is the modified TrecQA, and state-of-the-art refers to (Rao et al., 2016)

the Mean Reciprocal Rank (MRR) for evaluation purposes[5]. In contrast, the questions in the SQuAD are provided only with one right answer. For this reason, we use the Accuracy measure to evaluate the performance of the model. Please note that the Accuracy measure is the floor measure for the performance of this system though[6]. The experimental results are presented in Table 7.1.

The errors made by the system are of three types. Around 8% of mistakes are formed due to long sentences with nested structures for which the LSTMs in our system failed to generate correct or sufficient representations. Approximately 4% of the errors are due to unknown words for which we did not have word vectors. Since they scarcely occurred in the dataset, the algorithm couldn't provide them with accurate word vectors. The last 2% of errors are caused by syntactic problems such as incomplete sentences or typos in either questions or answers.

7.5 Conclusions

In this chapter, we presented a QA system with a state-of-the-art result in sentence-level answer extraction. The proposed architecture is designed to distinguish between good and bad sentence answers given a question. It is also designed to accept explicit textual features such as the common words between a question and an answer as a hard constraint in its objective function. It provides the architecture with more flexibility for applying heuristics specific to each use case.

Sentence selection can be considered either as a stand-alone QA system or as the first component for a fine-grained QA system. In the latter case, it enhances the precision and performance of the overall system by providing the correct

[5]Please see Section 2.4 for a detailed description of these measures.

[6]Compared to MRR and MAP, the accuracy is a pessimistic measure for this experiment because it just considers the first selected answer and disregards the others.

sentence hence decreasing the search space for the following components while in the first case it directly provides users with a coarse-grained answer.

We also showed that using models which are trained on large datasets and tuned on smaller datasets for other similar tasks (i.e., transfer learning) can be beneficial for QA. Designing architectures which can handle transfer learning is of prime importance, especially when dealing with small datasets or limited data. These architectures let us train our model with large datasets and tune and customize it with our little available data in the end. This model is also proved beneficial for other sentence-level inference tasks such as multi-lingual sentence alignment (Aghaebrahimian, 2018b).

8. Unstructured Question Answering

In this chapter, we describe our unstructured QA module for answering non-factoid questions from unstructured data. This module advocates a linguistically-based approach to answering non-factoid open-domain questions. First, we elaborate on an architecture for textual encoding based on which we introduce a deep end-to-end neural model. This architecture benefits from a bilateral attention mechanism which helps the model to focus on a question and the answer sentence at the same time for phrasal answer extraction. Second, we feed the output of a constituency parser into the model directly and integrate linguistic constituents into the network to help it concentrate on chunks of an answer rather than on its single words for generating more natural output. By providing a way to exploit syntax in the model we impose constraints on the candidate space, limit errors and make the system more efficient. By optimizing this architecture, we managed to obtain results near to human performance and competitive to a state-of-the-art system on SQuAD and MS-MARCO datasets respectively. The content of this chapter is taken from a paper (Aghaebrahimian, 2018a) which was written by the thesis author during his Ph.D. study and is published in the proceedings of the CoNLL 2018 conference.

8.1 Introduction

Reading, comprehending and reasoning over texts and answering a question about them is a fundamental aspect of computational intelligence. Question Answering (QA), as a measure of intelligence, has been even suggested to replace the Turing test (Clark and Etzioni, 2016).

The development of large datasets of QA in recent years (Hermann et al., 2015; Hill et al., 2015; Bordes et al., 2015; Rajpurkar et al., 2016) advanced the field especially for two significant branches of QA namely factoid (Aghaebrahimian and Jurčíček, 2016b,a) and non-factoid[1] (Rajpurkar et al., 2016) QA.

Non-factoid QA or QA over unstructured data is a somewhat new challenge in open-domain QA. A non-factoid QA system answers questions by reading

[1]While the answer to a non-factoid question is a chunk of one or more adjacent words, the answer to a factoid question is only an entity.

and comprehending a context. The context in which we assume the answer is mentioned may have different granularities from a single sentence or paragraph to larger units of text. A QA system is supposed to extract a phrase answer from the provided paragraph or sentence depending on its granularity level.

Text: The 2008 Summer Olympics torch relay was run from March 24 until August 8, 2008, prior to the 2008 Summer Olympics, with the theme of "one world, one dream". Plans for the relay were announced on April 26, 2007, in Beijing, China. The relay, also called by the organizers as the "Journey of Harmony", lasted 129 days and carried the torch 137,000 km (85,000 mi) – the longest distance of any Olympic torch relay since the tradition was started ahead of the 1936 Summer Olympics.

Question: Where were the details of the torch relay made known?
Sentence Answer: Plans for the relay were announced on April 26, 2007, in Beijing, China.
Phrase Answer: Beijing, China.

Box 8.1: Question Answering over unstructured data. The sample is taken from the SQuAD (Rajpurkar et al., 2016).

The context for answering questions is usually extracted using an Information Retrieval (IR) technique. Then, a QA system should extract the best answer sentence. There are many studies about extracting answer sentences including but not limited to (He et al., 2015; He and Lin, 2016; Yih et al., 2013; Yu et al., 2014; Rao et al., 2016; Aghaebrahimian, 2017a).

Extracting the final or shortest possible answer from a set of candidate answer sentences is addressed in many studies as well. (Zhang et al., 2017; Gong and Bowman, 2017; Shen et al., 2016; Weissenborn et al., 2017b). Instead of reasoning over and making inference on linguistic symbols (i.e., words or characters), almost all of these models use a neural architecture to encode contexts and questions into a vector representation and to reason over them.

A typical pattern in most of the current models is the use of a variant of uni- or bi-directional attention schemes (question to context and vice-versa) to encode the semantic content of questions' words with a focus on their context's words (Seo et al., 2016; Xiong et al., 2016; Weissenborn et al., 2017a; Chen et al., 2017; Wang et al., 2017). Compared to these models the novelty of our work is in explicitly conducting attention over both the context and the question for each candidate constituent[2] answer. The fact that this is better than attending only to the question words is investigated and proved according to the results reported in Section 8.6.

[2]From now on and for the sake of brevity by constituents we mean linguistic constituents as they are referred to in Phrase Structure Grammar (Noam Chomsky, 1957).

Constituents Type	Training set	Development set
NP	59 %	62 %
ROOT	8 %	6 %
NNP	5 %	4 %
NN	4 %	2 %
JJ	3 %	1 %
VP	3 %	4 %
CD	3 %	2 %
PP	2 %	4 %
S	2 %	2 %
others	11 %(each $< 2\%$)	13 %(each $< 2\%$)

Table 8.1: The distribution of constituent types of answers in the SQuAD training and development sets. For constituency parsing the Standford CoreNLP tool (Manning et al., 2014) is used. To see the full list of available constituency types in the dataset please refer to Appendix C.2

Another observation is that a majority of recent studies are purely based on data science where one can barely see a linguistic intuition towards the problem. We show that a pure linguistic intuition could help neural reasoning and attention mechanisms to achieve quantitatively and qualitatively better results in QA.

By analyzing a human-generated QA dataset called SQuAD (Rajpurkar et al., 2016), we realized that people tend to answer questions in units called constituents (please see Table 8.1). They expect an answer to a question to be a valid constituent otherwise it would probably not be grammatical.

Constituents and Constituency relations are the bases of Phrase Structure Grammar first proposed by Noam Chomsky (Noam Chomsky, 1957). Phrase Structure Grammar and many of its variants including Government and Binding theory (Chomsky, 1993) or Generalized and Head-driven Phrase Structure Grammar (Gazdar et al., 1994; Pollard and Sag, 1994) define hierarchical binary relations between the constituents of a text, and hence help to realize an exact and natural answer boundary for answer extraction.

Having these two points in mind and inspired by attentive pooling networks by Santos et al. (2014) we designed an attentive bilateral model and trained it on the constituents of questions and answers. We attempted to use some information from the parser, so to go beyond a simple word-based or vector-based representation as well as to show the relevance of linguistic constituents for answer extraction.

The results obtained by the model are near to human performance on SQuAD dataset and competitive to a state-of-the-art system on MS-MARCO dataset.

8.2 Constituency Relations

There is hardly a universal agreement upon the definition of the term 'constituent'. In general, a constituent is an inseparable unit that can appear in different places of a sentence. Instead of defining what a constituent is, linguists define a set of experiments such as replacement or expansion to distinguish between constituents and non-constituents.

For instance, let's consider the sentence 'Plans for the relay were announced on April 26, 2007, in Beijing, China.' We can replace or expand some constituents in Figure 8.1 and rephrase the sentence as 'on April 26, 2007, plans for the relay were announced, in Beijing, China.' or 'Plans for the great and important relay were announced on April 26, 2007, in Beijing, China.' while we are sure that these rephrases are not only both syntactically and semantically correct but also convey the same meaning as the original sentence. Zhang et al. (2017) and Xie and Eric (2017) are some of the earliest works which tried to integrate more linguistic structures into QA. Using TreeLSTM, Zhang et al. (2017) tried to incorporate linguistic structure into QA implicitly. At the prediction step, they used pointer network (Vinyals et al., 2017) to detect the beginning and the end of answer chunks. In contrast, Xie and Eric (2017) explicitly modeled candidate answers as sequences of constituents by encoding individual constituents using a chain of-trees LSTM (CT-LSTM) and tree-guided attention mechanism. However, their formulation of constituents is more complicated than ours, and as we will see, direct use of constituents as answer chunk is much less cumbersome and yields better results.

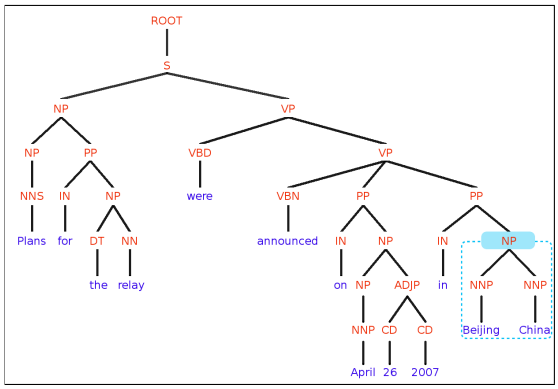


Figure 8.1: Constituency parse tree of the sentence 'Plans for the relay were announced on April 26, 2007, in Beijing, China.'. Blue tokens are surface words, and red tags are constituency types. The immediate tags attached to surface forms are word-level, and others are phrase level constituency types.

8.3 System Architecture

In this section, we describe how to represent questions, sentences and answers in vector space in Subsection 8.3.1 and then we train the vectors in Subsection 8.3.2 using a specific loss function and distance measure.

8.3.1 Representation Learning

Our goal is to extract constituent answers by loading their vector representations with the semantic content of their question and their containing answer sentence. To achieve this end, we integrated a bilateral attention mechanism into our model which lets us estimate a joint vector representation between answers when they are attending questions and when they are attending sentences.

To encode the semantic information in questions and sentences, we used a simple encoding unit (see Equations 8.1 to 8.6). In this unit, $W_k \in \mathbb{R}^{|V|}$ are words in one-hot vector representations where k^{th} element of each vector is one and others are 0. V are all vocabularies in questions and answers. $E \in \mathbb{R}^{|V| \times d_e}$ is the embedding matrix and d_e is the embedding dimension. The product of the multiplication in Equation 8.1 is the word embeddings in which each cell $W_{i,t}$ is the word in time step t in sample i. This is the input of forward and backward RNN cells in Equations 8.2 and 8.4.

As RNN cell, we used Long Short-Term Memory architecture (LSTM) (Hochreiter and Schmidhuber, 1997). Hu et al. (2017) and Pan et al. (2017) show that bi-directional LSTM architectures provide more accurate representations of textual data. The common practice to form a bidirectional LSTM is to concatenate the last vectors in forward and backward LSTMs. Instead, we used a stepwise max pooling (STMP) mechanism which takes the most essential vectors from the forward and backward LSTMs in Equations 8.3 and 8.5 and concatenate them in Equations 8.6.

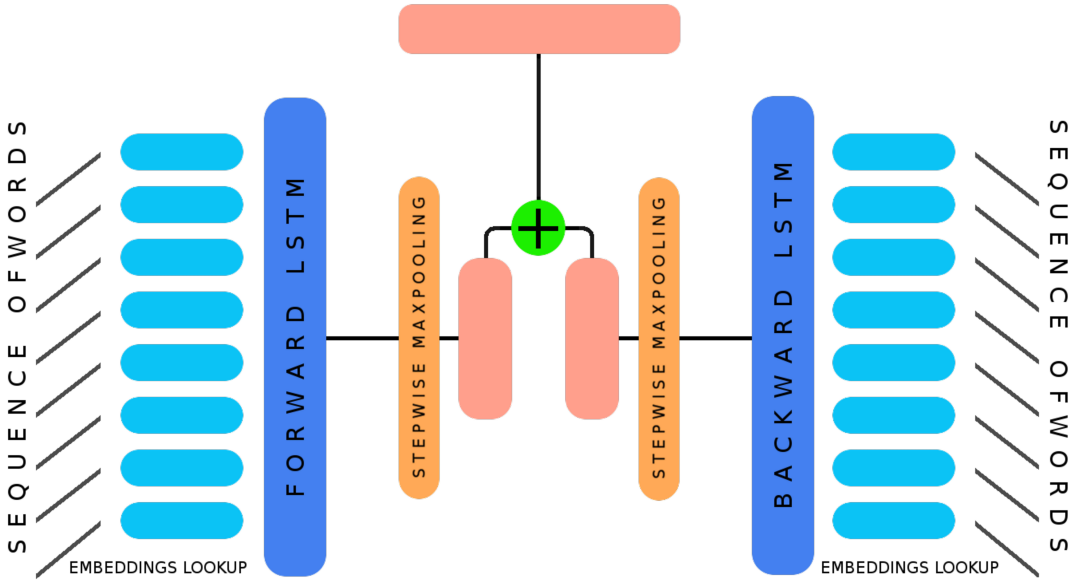


Figure 8.2: The encoding unit. The embedding lookup uses pre-trained Glove word vectors (Pennington et al., 2014a) and updates them through training. The output is the concatenation of max-pooled vectors of LSTM encoders.

$$W_{i,t} = E^{\top} W_k \tag{8.1}$$

$$\overrightarrow{enc}_{i,t} = LSTM(\overrightarrow{enc}_{i,t-1}, W_{i,t}) \tag{8.2}$$

$$\overrightarrow{enc}_{i} = SWMP(\overrightarrow{enc}_{i,t}) \tag{8.3}$$

$$\overleftarrow{enc}_{i,t} = LSTM(\overleftarrow{enc}_{i,t+1}, W_{i,t}) \tag{8.4}$$

$$\overleftarrow{enc}_{i} = SWMP(\overleftarrow{enc}_{i,t}) \tag{8.5}$$

$$enc_i = [\overrightarrow{enc}_i; \overleftarrow{enc}_i] \tag{8.6}$$

Using our encoding unit (see Figure 8.2) we encode questions and sentences and then concatenate the resulted vectors to generate a joint representation of questions and their answer sentences in Equation 8.7.

$$enc_i^{QS} = [enc_i^{Q}; enc_i^{S}] \tag{8.7}$$

$$W_{i,t}^{A} = E^{\top} W_{k}^{A} \tag{8.8}$$
$$\overrightarrow{h_{i,t}} = LSTM(\overrightarrow{h_{i,t-1}}, W_{i,t}^{A}) \tag{8.9}$$
$$\overrightarrow{h_{i,t}^{A_Q}} = ATT(\overrightarrow{h_{i,t}}, \mathbf{enc_i^Q}) \tag{8.10}$$
$$\overrightarrow{h_{i}^{A_Q}} = SWMP(\overrightarrow{h_{i,t}^{A_Q}}) \tag{8.11}$$
$$\overleftarrow{h_{i,t}^{A_Q}} = LSTM(\overleftarrow{h_{i,t+1}}, W_{i,t}^{A}) \tag{8.12}$$
$$\overleftarrow{h_{i,t}^{A_Q}} = ATT(\overleftarrow{h_{i,t}^{A_Q}}, \mathbf{enc_i^Q}) \tag{8.13}$$
$$\overleftarrow{h_{i}^{A_Q}} = SWMP(\overleftarrow{h_{i,t}^{A_Q}}) \tag{8.14}$$
$$h_{i}^{A_Q} = [\overrightarrow{h_{i}^{A_Q}}; \overleftarrow{h_{i}^{A_Q}}] \tag{8.15}$$

In the next step, we need to encode the constituent answers. Our answer encoding unit has two modules, one with attention on questions enc_i^Q (Equations 8.8 - 8.15) and the other with attention on sentences enc_i^S (Equations 8.16 - 8.23). In both modules, we used an architecture similar to the one in the encoding unit with an additional attention unit.

In the answer encoding unit, again the input to LSTM cells are word embeddings generated by the lookup table $W_{i,t}^A$. Two attention layers in this unit receive the output sequences of the forward and backward LSTM cells and focus once on questions and once on sentences.

$$W_{i,t}^{A} = E^{\top} W_{k}^{A} \tag{8.16}$$
$$\overrightarrow{h_{i,t}} = LSTM(\overrightarrow{h_{i,t-1}}, W_{i,t}^{A}) \tag{8.17}$$
$$\overrightarrow{h_{i,t}^{A_S}} = ATT(\overrightarrow{h_{i,t}}, \mathbf{enc_i^S}) \tag{8.18}$$
$$\overrightarrow{h_{i}^{A_S}} = SWMP(\overrightarrow{h_{i,t}^{A_S}}) \tag{8.19}$$
$$\overleftarrow{h_{i,t}^{A_S}} = LSTM(\overleftarrow{h_{i,t+1}}, W_{i,t}^{A}) \tag{8.20}$$
$$\overleftarrow{h_{i,t}^{A_S}} = ATT(\overleftarrow{h_{i,t}^{A_S}}, \mathbf{enc_i^S}) \tag{8.21}$$
$$\overleftarrow{h_{i}^{A_S}} = SWMP(\overleftarrow{h_{i,t}^{A_S}}) \tag{8.22}$$
$$h_{i}^{A_S} = [\overrightarrow{h_{i}^{A_S}}; \overleftarrow{h_{i}^{A_S}}] \tag{8.23}$$

In the end, the vectors generated by these two modules are concatenated[3] to form a general attentive representation of constituents with respect to their corresponding questions and sentences.

$$h_i^{A_{QS}} = [h_i^{A_Q}; h_i^{A_S}] \tag{8.24}$$

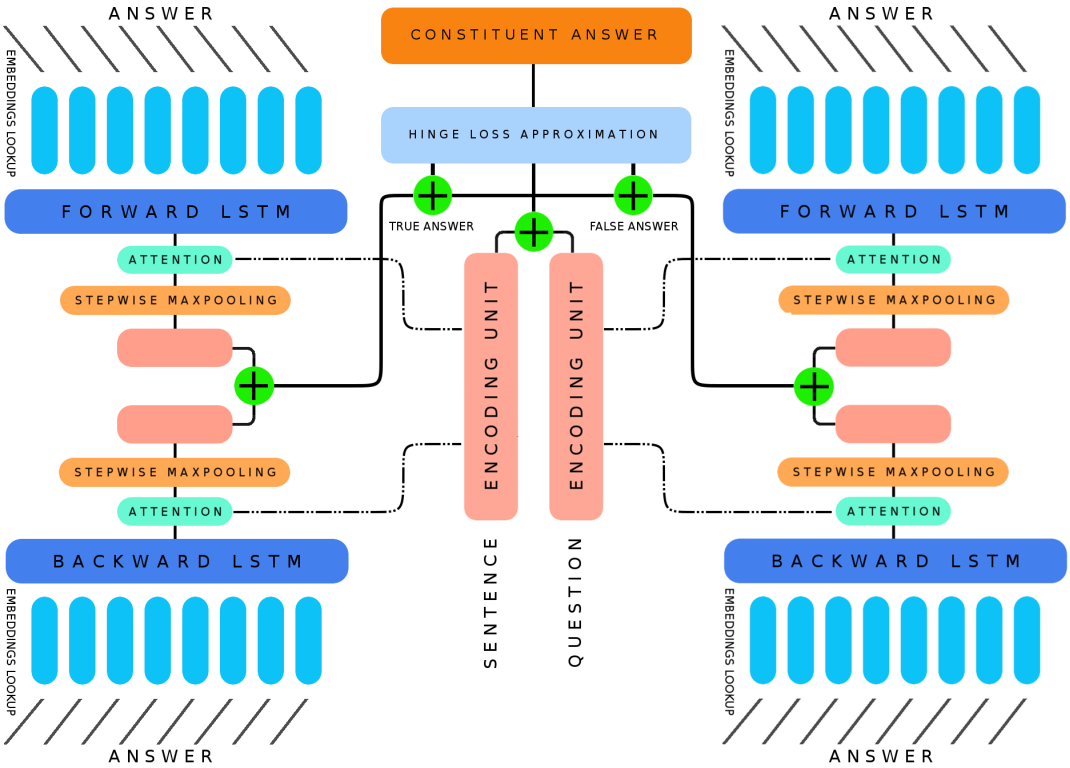


Figure 8.3: The system architecture. The encoding units are illustrated in Figure 8.2. Two answer modules, one with attention on questions and the other on sentences, provide a joint representation containing all required information with respect to questions and sentences for making inference on true constituents. The Hinge loss function accepts three inputs, joint representation of question and answer sentence, true answer representation and false answer representation both with attention to respective question and sentence. Given each question and a series of constituents, it generates a score for each constituent and returns their argmax as the final answer.

At training time, we try to learn the vector representations of questions, sentences, and their constituents jointly. However, we like to learn the vectors in a way that leads to a small distance between questions and their true constituents and a long distance between them and their false constituents. For this purpose,

[3]All concatenations are performed on the last layer (i.e., data dimensions).

for each pair of question and sentence, we compute a true answer A^+_{QS} and false answer A^-_{QS} vectors.

These vectors are generated by passing a correct constituent A^+ and a random wrong constituent A^- through question-attentive (Equations 8.8 - 8.15) and sentence-attentive (Equations 8.16 - 8.23) modules and by concatenating the outputs.

8.3.2 Training

In this section, we train our model. Given a question and the constituents associated with its answer sentence, the model generates a score for each constituent. The score is an estimate of how similar the constituent to the gold answer is. The model returns the id of its true predicted constituent by taking the argmax over the scores.

To train the model, we need to compute the distance between questions and their true constituents and to contrast it with the distance between questions and their false constituents.

There are various measures of distance or similarity between two vectors each with its own merits. Feng et al. (2015a) did an exhaustive study on different distance measures for text classification and proposed some new measures including the Geometric mean of Euclidean and Sigmoid Dot product (GESD) (following equation) which outperformed other measures in his study[4].

$$DIS(Q, A) = \frac{1}{1 + exp(-(Q.A))} * \frac{1}{1 + ||Q - A||}$$

We integrated GESD in our work to estimate the distance between questions and their true and false constituents. GESD linearly combines two other measures called L2-norm and inner product. L2-norm is the forward-line semantic distance between two sentences and inner product measures the angle between two sentence vectors.

Now everything is ready to train the model. The overall system architecture is illustrated in Figure 8.3. We use the Hinge loss to estimate the loss of each

[4]For an ablation study on the effect of various distance measures please see Appendix C

question-answer combination. The Hinge function increases the loss with the distance between a question and its true constituents while decreases it with the distance between a question and its false constituents. In our Hinge function as seen in the following equation, enc^{QS} is the joint vector representation of questions and their answer sentences, $h_i^{A^-_{QS}}$ is the vector of wrong answers, $h^{A^+_{QS}}$ is the vectors of right answers. Finally, m is the margin between positive and negative answers. It makes a trade-off between the mistakes in positive and negative classes.

$$\mathcal{L} = \sum_i \max(0, m + DIS(enc_i^{QS}, h_i^{A^+_{QS}}) - DIS(enc_i^{QS}, h_i^{A^-_{QS}}))$$

8.4 Dataset

We used the SQuAD to evaluate the performance of our model[5]. The answers in the SQuAD can be any span of adjacent words and are guaranteed to be existent in the paragraph which comes with each question. These answers are categorized into ten types including Person, Date, Location, etc (Rajpurkar et al., 2016). However, there are no statistics available on the constituent type of each answer.

To pre-process the questions and sentences in the dataset, we removed all non-alphanumeric characters from all contents. Then, we replaced numeric values with '9' since we needed to control the vocabulary size by eliminating redundant numeric values, but at the same time, we wanted to parse the contents, and we needed to keep the semantic values of numeric tokens. Then we used CoreNLP tool (Manning et al., 2014) to tokenize and to perform constituency parsing on the contents.

After extracting constituents from the tree of sentences and comparing them with gold answers, we realized that 72% of the responses are constituents. Other 16% of the responses had slight divergences from a constituent, like lacking or having an extra determiner or punctuation mark which were eventually going to be disregarded in the official evaluation script. The remaining 12% was a combination of two smaller constituents or a part of a larger one.

In the training set, to use constituents as answers, we replaced non-matching responses with the smallest and most similar constituents. Since at the evaluation time, we needed the gold answers and not their replaced constituents, we didn't change the answers in the development set.

[5]For more about this dataset, please see Section 4.4

Total number of 48 different constituents including both terminal and non-terminal ones are extracted from the SQuAD. The percentage of each constituent type in training and development sets are presented in Table 8.1. The figures for the development set are computed only based on exact match answers.

We used SQuAD's development set for testing the system and reporting the results. To prepare the dataset for training and evaluating our system we used a state-of-the-art answer sentence selection system Aghaebrahimian (2017a) to extract the best answer sentences. The system provided us with the best sentence with 94.4 % accuracy given each question. After pre-processing the sentenc as explained above, we extracted its constituents and trained the model using the correct constituents as true and other constituents as negative samples.

For evaluation purpose, we used the SQuAD's official evaluation script which computes the Exact Match and F1. The Exact Match is the percentage of predictions which exactly match the gold answer and F1 (Macro-averaged) is the overlap between predictions and gold answers[6].

To perform more experiments with other datasets, we tried our model on MS-MARCO dataset (Nguyen et al., 2015) too. As a machine comprehension dataset, MS-MARCO has two fundamental differences with SQuAD. Every question in MS-MARCO has several passages from which the best answer should be retrieved. Moreover, the responses in MS-MARCO are not necessarily sub-spans of the provided contexts so that BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004) should be used as the metrics for evaluation. These are the two metrics which are used in the official MS-MARCO evaluation tool. During training we used the highest BLEU scored constituent as the answer and in the assessment, we computed the BLEU and ROUGE scores of the constituents selected by the system. As the results in Table 8.2 show, our system obtained competitive results to another state-of-the-art system trained on the same dataset.

8.5 Experiment

To evaluate our new architecture and to see how integrating linguistic constituents affects its performance we set up two settings. We designed one setting for assessing the effect of using constituents instead of words (constituent-base vs. word-base) and another to evaluate the impact of using attention mechanism on top of vector training modules (uni- vs. bi-attention). Therefore we conducted

[6]For more about the evaluation metrics please refer to Section 2.4

four experiments on both datasets or eight experiments in total.

In the constituent-base setting, we generated the training and test data as described in Section 8.4. In the word-base setting, however, we replaced constituents with the tokens in answer sentences for both train and test sets and trained our model to compute two scores for initial and final positions of answer chunks. In the constituent-base setting at test time, we directly used the predicted constituent as the final answer. In the word-base setting, however, we got the final answer using the highest-scored words for initial and final positions.

For training our model, we used 300-dimensional pre-trained Glove word vectors (Pennington et al., 2014b) to generate the embedding matrix and kept the embedding matrix updated through training. We used 128-dimensional LSTMs for all recurrent networks and used 'Adam' with parameters learning rate=0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$ for optimization. We set batch size to 32 and dropout rate to 0.5 for all LSTMs and embedding layers. We performed the accuracy check only on the first best answer.

We also investigated the effect of bilateral attention on the model performance. In the bi-attention model, we used the model as described in Section 8.3. In the uni-attention model, we eliminated the attention on sentences and only used the module for attention on questions. The details and statistics of the experiments are presented in Section 8.6.

Using the early-stopping technique for training, we run all models for ten epochs. The learning curves of the models are illustrated in Figure 8.4.

8.6 Results

The results of our experiments are summarized in Table 8.2. By contrasting the results of uni-/bi-attention and word/constituent-base models, we can see that the proposed bi-attention mechanism with linguistic constituents integrated into it makes a significant improvement on answer extraction. Another interesting observation is that the Exact Match metric benefits from restriction to constituents as answers. Concerning the MS-MARCO dataset, the results are competitive to the state-of-the-art system on the same dataset (Wang et al., 2017)
.

We did a study on the learning capacity of the model as well. As the learning

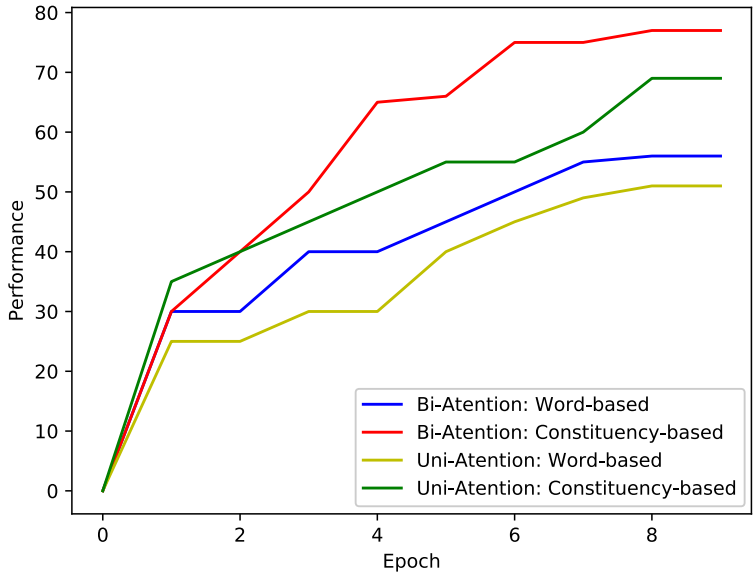


Figure 8.4: The learning curves as a function of the number of epochs.

curves in Figure8.5 illustrate, the model saturated after around 60,000 training sentences, which suggests that a larger model with more parameters can make use of extra available training samples. We intend to pursue this objective in our future work.

	SQuAD Development set		MS-MARCO Evaluation set	
	Exact-match(%)	F1(%)	BLEU	ROUGE
Logistic Regression (Rajpurkar et al., 2016)	40.00 %	51.00 %	-	-
Uni-Attention Word-base (this work)	55.12 %	57.98 %	35.6	35.1
Bi-Attention Word-base (this work)	59.84 %	63.08 %	38.1	38.4
Uni-Attention Constituency-base (this work)	73.82 %	77.43 %	39.6	39.9
TreeLSTM (Zhang et al., 2017)	69.10 %	78.38 %	-	-
BIDAF (Seo et al., 2016)	72.6 %	80.7 %	-	-
CCNN (Xie and Eric, 2017)	74.1 %	82.6 %	-	-
R-net (Wang et al., 2017)	75.60 %	82.80 %	42.2	42.9
Bi-Attention Constituency-base (this work)	**80.72 %**	**83.25 %**	42.1	42.7
Human Performance (Rajpurkar et al., 2016)	82.30 %	91.22 %	-	-

Table 8.2: The performances of different models in the exact match/F1 metrics for the SQuAD and BLEU/ROUGE for the MS-MARCO dataset.

8.7 Ablation and Error Analysis

In this section, we analyze the SQuAD concerning answer distribution over different query types. Table 8.1 shows that the NP type constituents are the most prominent type among all other answers. However, to investigate the importance of other types in overall system performance, we performed an ablation study where we studied the influence of each constituent type on overall accuracy. The

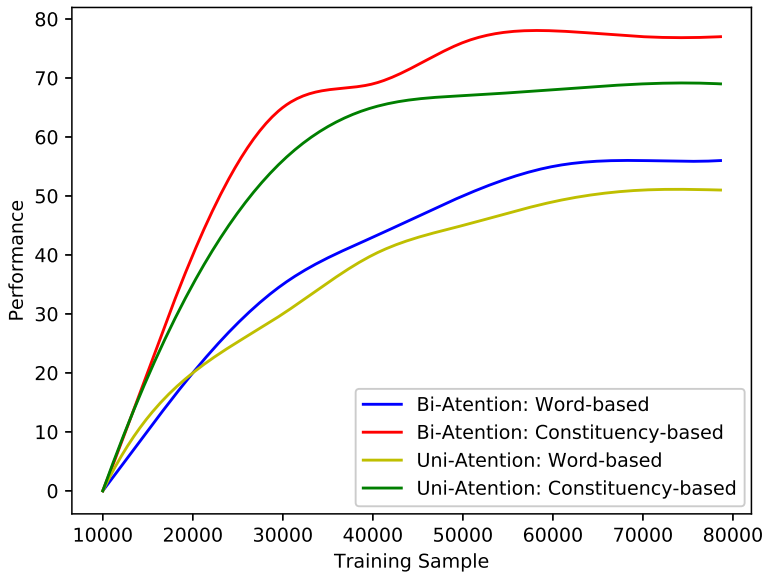


Figure 8.5: The learning curve as a function of the number of training samples. After around 60,000 samples the models didn't improve anymore.

results are presented in Figure 8.6. We also examined how much the model succeeded in retrieving answers from each type. The results are presented in the same figure. This table also shows how often does the method succeed in cases where the correct answer is, in fact, a constituent span.

As seen in Figure 8.6, the answers are mostly singular noun phrases after which with a significant difference are proper nouns, verb phrases, and prepositional phrases. We can also see how the model performed for each constituent. It seems that extracting cardinal numbers is much easier for the model than retrieving roots or full sentences.

An analysis of the errors shows that false answer sentence, non-constituent answers, parsing errors, overlapping constituents and unknown words are the primary reasons for the mistakes made by our system. The sentence selection process brought about six percent incorrect answers. The next primary reason for making mistakes is the constituents which contain other smaller constituents. While in all cases we extract the smallest constituent, in about three percent of overlapping constituents the more extended ones are the correct answer. Parsing errors where the constituents are not retrieved correctly and unknown words where the embeddings are not appropriately trained are responsible for other four percent of the mistakes. Finally, non-constituent answers led to around eight percent false answers in the system output.

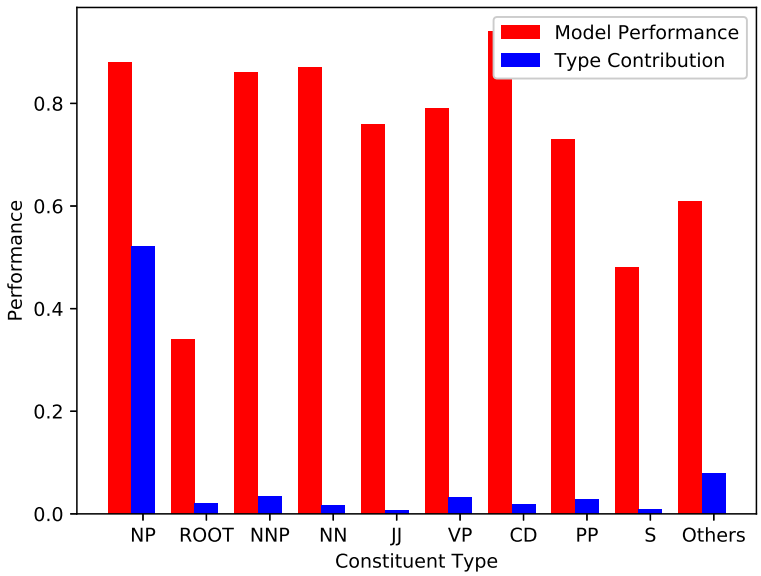


Figure 8.6: Blue lines are the contribution of each type in the overall system performance using the best model and at the convergence time. Red bars represent the performance of the model in retrieving each constituent type when the model is converged. Performance is expressed in the exact match metric (%). As a guide to how to read the chars, the first blue line for NP type says that 50% of all correctly extracted answers by the system are NP type-answers. The red line of the same type says that our system managed to retrieve about 87% of all NP-type answers in the dataset.

8.8 Conclusions

We described a new linguistically-based end-to-end DNN for Question Answering from unstructured data. This model is a neural formulation in which linguistic constituents are explicitly modeled. It operates an LSTM over the constituents and uses the resulting hidden states to attend both to question and to the encompassing context sentence, thereby enriching the constituent's representation with both.

We successfully integrated some linguistic information from the parser into the network, so to go beyond a simple word-based or vector-based representation as well as to show the relevance of linguistic constituents for answer extraction.

The use of constituents instead of an arbitrary string of words in answers improves the system performance in several ways. First, it increases the precision of the system. By looking at the small gap between the F1 and the exact match metrics in our system and compare it to the ones for the other systems, we can

see that the ratio of exact-match answers in our system is higher than that of the other ones. It says that the answers extracted in this system has relatively higher level of precision.

Second, it helps an answer to look more like a human-generated one. Considering prediction and ground truth as bags of tokens, the F1 (Macro-averaged) metric computes the average overlap between the prediction and ground truth answer. While a predicted response may have a full overlap with the ground truth hence gain a high F1 score, due to the irrelevant words, it contains, it poses an incoherent answer to users. Users generally expect an answer to be a valid constituent otherwise it would probably not be grammatical. The longer the gap between exact match and F1 measures, the more irrelevant words appear in responses. This is primarily an essential factor in the overall quality of dialogue QA systems where users expect to receive a natural and human-generated-like answer.

Last but not least, imposing constraints on the candidate space, limit errors and make the system more efficient by decreasing the search space and weeding out non-relevant answers. In the future, we plan to integrate dependency relations into the model by designing a larger model and to evaluate it on other QA datasets.

9. Conclusion

The goal we pursued in this thesis was to advance the state of the art in Question Answering by designing and integrating new machine learning technologies with an emphasis on Deep Neural Networks in QA. Based on the results of our work we believe that we accomplished this goal successfully.

The theoretical foundations of Question Answering are collected in this thesis. The first chapter gives the readers an overview of the field while the second chapter presents them a fundamental understanding of the techniques and technologies which are going to be used in further chapters. An in-depth overview of QA systems is presented in chapter three. Datasets in QA are introduced in chapter four as an introduction to chapter five which describes the Q2AD, one of the contributions of this work.

In chapter six the readers are introduced to a structured QA system based on a knowledge graph and three different machine learning models namely Logistic Regression, Neural Network, and Convolution Neural Network. In chapter seven a Deep Neural Network based on Long Short-Term Memory cells is presented for sentence-level answer extraction and in chapter eight a bi-attention Deep Neural Network enhanced by constituency relations is described. In these last three chapters, the readers get an in-depth presentation of three different QA subsystems which make up our hybrid QA system.

A hybrid QA system and a dataset for sentence- and word-level QA, in addition to several publications in different international NLP conferences whose the leading researcher is this thesis author are the main contributions of this work.

In this work, we presented a hybrid system which integrates knowledge graphs and free texts for answering open-domain structured and unstructured questions. To the best of our knowledge, this is the first fully neural system incorporating knowledge graphs and free texts as a hybrid QA system. All modules in this system benefit from DNNs, due to the state-of-the-art performance of them not only in our system but also in many other NLP tasks. This system is composed of a structured entity-level, an unstructured sentence-level, and an unstructured word-level QA subsystems all with state-of-the-art results.

In the structured QA subsystem, we introduced a highly scalable approach for open-domain structured or factoid QA. In contrast to many similar factoid QA systems which work on limited versions of the Freebase, our system seamlessly

works with a live and full version of the Freebase which makes it able to extract the most relevant and up-to-date answers given any time.

In the unstructured sentence-level QA subsystem, we proposed the Constrained Deep Neural Network (CDNN) which can enforce hard constraints on the parameters of a Deep Neural Network for sentence answer selection. The CDNN makes its predictions based on the neural reasoning compound with some symbolic constraints. We also presented a successful case of transfer learning technique on a model with large learning capacity and showed that it could obtain excellent results without dependence on any external resources or doing any time-consuming feature designing procedure.

And in the unstructured word-level QA subsystem, we proposed a new approach to answering open-domain questions from unstructured data. In this approach, we introduced a deep end-to-end neural model which benefits from a bilateral attention mechanism. This architecture can focus on a question and the answer sentence at the same time for answer extraction. Then, we integrated constituency relations into the network to help it concentrate on chunks of an answer rather than on its single words.

We also introduced Quora Question Answer Dataset (Q2AD). The Q2AD is composed of questions which are posed in Quora Question Answering site. It is the only dataset which provides multiple-part sentence-level and word-level answers. Moreover, the questions in the dataset are authentic which is much more realistic for real-life Question Answering systems.

For the future of this work, the author is seeking forward integrating more challenging components such as complex QA or mathematical QA as separate modules into the system as well as crowdsourcing the Q2AD for compiling a large scale dataset.

To conclude, the main contribution of the thesis is thorough research and developing experimental systems with significant empirical results in the three most prevalent and challenging QA types, namely;

- structured factoid,
- unstructured sentence-level, and
- unstructured word-level Question Answering.

Acknowledgments

The Ministry of Education, Youth and Sports of the Czech Republic partially funded this research under the grant agreement LK11221, core research funding, SVV project numbers 260 333 and 260 453, and GAUK 207-10/250098 of Charles University in Prague. This work has also been using language resources distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2010013). The author gratefully appreciates the thesis reviewers as well as his supervisor for their helpful comments on the final draft. He is also thankful to anonymous annotators for doing a great job in the Q2AD compilation.

Bibliography

Ahmad Aghaebrahimian. Constraint-based Semantic Parsing. In *proceeding of the Week of Doctoral Studies, Charles University, Faculty of Mathematics and Physics*, 2015.

Ahmad Aghaebrahimian. Constrained Deep Answer Sentence Selection. In *Proceedings of the 20th International Conference on Text, Speech, and Dialogue (TSD)*, 2017a.

Ahmad Aghaebrahimian. Quora Question Answer Dataset. In *20th International Conference on Text, Speech, and Dialogue (TSD)*, 2017b.

Ahmad Aghaebrahimian. Hybrid Deep Open-Domain Question Answering. In *Proceedings of the 8th Language and Technology Conference (LTC)*, 2017c.

Ahmad Aghaebrahimian. Linguistically-Based Deep Unstructured Question Answering. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL 2018)*, Brussels, Belgium, 2018a.

Ahmad Aghaebrahimian. Deep Neural Networks at the Service of Multilingual Parallel Sentence Extraction. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING 2018)*, Santa Fe, New-Mexico, USA, 2018b.

Ahmad Aghaebrahimian and Filip Jurčíček. Machine Learning for Semantic Parsing in Review. In *Proceedings of the 7th Langauge and Technology Conference (LTC), Poznan, Poland*, 2015a.

Ahmad Aghaebrahimian and Filip Jurčíček. Constraint-based Statistical Spoken Dialogue Systems. In *Proceedings of the young researcher roundtable in spoken dialog (YRRSDS)*, 2015b.

Ahmad Aghaebrahimian and Filip Jurčíček. Open-domain Factoid Question Answering via Knowledge Graph Search. In *Proceedings of the Workshop on Human-Computer Question Answering, The North American Chapter of the Association for Computational Linguistics (NAACL)*, 2016a.

Ahmad Aghaebrahimian and Filip Jurčíček. Constraint-Based Open Domain Question Answering Using Knowledge Graph Search. In *Proceedings of the 19th International Conference on Text, Speech and Dialogue (TSD), LNAI 9924*, 2016b.

I. Androutsopoulos, G. Ritchie, and P. Thanisch. Natural language interfaces to databases, an introduction. *Natural Language Engineering*, 1995.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. The sixth pascal recognizing textual entailment challenge. *Text Analysis Conference (TAC)*, 2008.

Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Proceedings of Association for Computational Linguistics (ACL)*, 2014.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

Kurt Bollacker, Patrick Tufts, Tomi Pierce, and Robert Cook. A Platform for Scalable, Collaborative, Structured Information Integration. In *Proceedings of the Sixth International Workshop on Information Integration on the Web*, 2007.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*, 2015.

Qingqing Cai and Alexander Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of of Association for Computational Linguistics (ACL)*, 2013.

Bob Carpenter. *Type-Logical Semantics*. The MIT Press, 1997.

Ming-Wei Chang, Dan Goldwasser, Dan Roth, and Vivek Srikumar. Discriminative learning over constrained latent representations. In *Proceedings of The North American Chapter of the Association for Computational Linguistics (NAACL)*, 2010.

Ming-Wei Chang, Lev Ratinov, and Dan Roth. Learning with Constrained Conditional Models. *Machine Learning*, page 399–431, 2012.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprintarXiv:1704.00051*, 2017.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of the Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

Noam Chomsky. *Lectures on Government and Binding: The Pisa Lectures*. Mouton de Gruyter, 1993.

Kenneth Ward Church and Patrick Hanks. Word Association Norms, Mutual Information, and Lexicography. *Computational Linguistics*, 16(1), March 1990. ISSN 0891-2017.

Peter Clark and Oren Etzioni. My computer is an honor student but how intelligent is it? standardized tests as a measure of AI. *AI Magazine*, 2016.

Peter Clark, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Turney, and Daniel Khashabi. Combining retrieval, statistics, and inference to answer elementary science questions. In *Proceedings of the 30th conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, 2016.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving Semantic Parsing from the World's Response. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, 2010.

Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (almost) from Scratch. *Machine Learning Research*, 2011.

Ann Copestake and Karen Sparck Jones. Natural language interfaces to databases. *Knowledge Engineering Review*, 1989.

Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-Attention Neural Networks for Reading Comprehension. *arXiv preprint arXiv:1607.04423*, 2016.

Zihang Dai, Lei Li, and Wei Xu. CFO: Conditional focused neural question answering with large-scale knowledge bases. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2016.

Freebase data dumps. `https://developers.google.com/freebase/`, 2015. Accessed: 2015-09-30.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. Question answering over Freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL/JCNLP)*, 2015.

Susan Dumais, Michele Banko, Eric Brill, Jimmy Lin, and Andrew Ng. Web question qnswering: Is more always better? In *Proceedings of the 25th annual international conference of ACM Special Interest Group on Information Retrieval (SIGIR)*, 2002.

Jeffry L. Elman. Finding structure in time. *Cognitive Science*, 14(2), 1990.

Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

Minwei Feng, Bing Xiang, Michael R. Glass, Lidan Wang, and Bowen Zhou. Applying deep learning to answer selection: a study and an open task. In *Proceedings of IEEE ASRU Workshop*, 2015a.

Minwei Feng, Bing Xiang, Michael R. Glass, Lidan Wang, and Bowen Zhou. Applying deep learning to answer selection: A study and an open task. In *Proceedings of IEEE ASRU workshop*, 2015b.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya a. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. Building Watson: An Overview of the DeepQA Project. *AI Magazine*, 31(3), 2010.

Gerald Gazdar, Ewan H. Klein, Geoffrey K. Pullum, and Ivan A. Sag. *Generalized Phrase Structure Grammar*. Blackwell, Oxford, 1994.

Dan Goldwasser and Dan Roth. Learning from natural instructions. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

Yichen Gong and Samuel R. Bowman. Ruminating reader: Reasoning with gated multi-hop attention. *arXiv preprint arXiv:1704.07415*, 2017.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 2014.

Bert F. Jr. Green, Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question answerer. *Massachusetts Institute of Technology-Lincoln Laboratory*, 1961.

Sven Hartrumpf, Ingo Glöckner, and Johannes Leveling. Efficient question answering with question decomposition and multiple answer streams. In *proceedings of the conference on Evaluating Systems for Multilingual and Multimodal Information Access*, 2009.

Hua He and Jimmy Lin. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2016.

Hua He, Kevin Gimpel, and Jimmy Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, 2015.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv :1511.02301*, 2015.

Lynette Hirschman, Marc Light, Eric Breck, and John D. Burger. Deep read: A reading comprehension system. In *Proceedings of the Association for Computational Linguistics (ACL)*, 1999.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), 1997.

Eduard Hovy, Laurie Gerber, Ulf Hermjakob, Michael Junk, and Chin-Yew Lin. Question answering in webclopedia. In *Proceedings of the Tenth Text REtrieval Conference (TREC-2001)*, 2001.

Minghao Hu, Yuxing Peng, and Xipeng Qiu. Reinforced mnemonic reader for machine comprehension. *arXiv preprint arXiv:1705.02798*, 2017.

Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Tomasz Jurczyk, Michael Zhai, and Jinho D. Choi. SelQA: A New Benchmark for Selection- based Question Answering. In *Proceedings of the 28th International Conference on Tools with Artificial Intelligence*, 2016.

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2016.

Aditya Kalyanpur, Siddharth Patwardhan, Branimir Boguraev, Adam Lally, and Jennifer Chu-Carroll. Fact-based question decomposition for candidate answer re-ranking. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011.

Fereshte Khani, Martin Rinard, and Percy Liang. Unanimous prediction for 100semantic mappings. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2016.

Daniel Khashabi, Tushar Khot, Ashish Sabharwal, Peter Clark, Oren Etzioni, and Dan Roth. Question Answering via Integer Programming over Semi-Structured Knowledge. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jeongwoo Ko, Luo Si, and Eric Nyberg. A probabilistic graphical model for joint answer ranking in question answering. In *Proceedings of the 30th ACM Special Interest Group on Information Retrieval (SIGIR)*, 2007.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of The 33rd International Conference on Machine Learning*, page 1378–1387, 2016.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2010.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, 1998.

Percy Liang. Lambda Dependency-based Compositional Semantics. *arXiv preprint arXiv:1309.4408*, 2013.

Christian Liguda and Thies Pfeiffer. A question answer system for math word problems. In *Proceedings of the First International Workshop on Algorithmic Intelligence*, 2011.

Chin-Yew Lin. ROUGE: a Package for Automatic Evaluation of Summaries. In *Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004)*, 2004.

Jimmy Lin and Boris Katz. Question answering from the web using knowledge annotation and knowledge mining techniques. In *Proceedings of the twelfth international conference on Information and knowledge management, ACM*, page 116–123, 2003.

Bernardo Magnini, Matteo Negri, Roberto Prevete, and Hristo Tanev. Mining knowledge from repeated cooccurrences: DIOGENE at TREC-2002. In *Proceedings of the Eleventh Text REtrieval Conference (TREC-2002)*, 2002.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, 2014.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to Parse Natural Language Commands to a Robot Control System. In *Proceedings of the 13th International Symposium on Experimental Robotics*, 2012.

Ana Cristina Mendes and Luisa Coheur. An approach to answer selection in question-answering based on semantic relations. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukas Burget, and Jan Honza Cernocky. RNNLM - Recurrent Neural Network Language Modeling Toolkit. In *Proceedings of the 2011 ASRU Workshop*, 2011.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of ICLR Workshop*, 2013.

Alessandro Moschitti and Silvia Quarteroni. Linguistic kernels for answer reranking in question answering systems. In *Proceedinds of the conference on Information Processing and Management*, 2010.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human generated machine reading comprehension dataset. *CoRR, abs/1611.09268*, 2015.

Noam Chomsky. *Syntactic structures*. The Hague, Paris: Mouton, 1957.

Boyuan Pan, Hao Li, Zhou Zhao, Bin Cao, Deng Cai, and Xiaofei He. MEMEN: Multi-layer embedding with memory networks for machine comprehension. *arXiv preprint arXiv:1707.09098*, 2017.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual meeting of the Association for Computational Linguistics (ACL)*, 2002.

Pavel Pecina. *Lexical Association Measures: Collocation Extraction*. Institute of Formal and Applied Linguistics, 2008.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014a.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Procedings of the conference Empirical Methods in Natural Language Processing (EMNLP)*, 2014b.

A. V. Phillips. A Question-Answering Routine. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1960.

Carl Pollard and Ivan A. Sag. *Head-driven phrase structure grammar*. University of Chicago Press, Chicago, 1994.

John Prager, Eric Brown, and Anni Coden. Question answering by predictive annotation. In *Proceedings of the ACM Special Interest Group on Information Retrieval (SIGIR)*, 2000.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. Natural language inference via dependency tree mapping: An application to question answering. In *Computational Linguistics*, 2004.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv preprint arXiv:1606.05250*, 2016.

Jinfeng Rao, Hua He, and Jimmy Lin. Noise-Contrastive Estimation for Answer Selection with Deep Neural Networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, 2016.

Deepak Ravichandran and Eduard Hovy. Learning surface text patterns for a question answering system. *Association for Computational Linguistics (ACL)*, 2002.

Matthew Richardson. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

Dan Roth and Wen-tau Yih. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, 2004.

Cicero dos Santos, Ming Tan, Bing Xiang, and Bowen Zhou. Attentive pooling networks. *arXiv preprint arXiv:1602.03609v1*, 2014.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

Dan Shen and Dietrich Klakow. Exploring correlation of dependency relation paths for answer extraction. *International Conference on Computational Linguistics and Association for Computational Linguistics (COLING/ACL)*, page 889–896, 2006.

Dan Shen and Dietrich Klakow. Exploring correlation of dependency relation paths for answer extraction. *Proceedings of the 21st International Conference on Computational Linguistics*, 2015.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016)*, 2016.

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, 2005.

Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the Advances in Neural Information Processing Systems*, 2013.

Vivek Srikumar and Dan Roth. A joint model for extended semantic role labeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, 2007.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-End Memory Networks. In *Advances in Neural Information Processing Systems 28*, 2015.

Md Arafat Sultan, Vittorio Castelli, and Radu Florian. A joint model for answer sentence ranking and answer extraction. *Transactions of the Association for Computational Linguistics (ACL)*, 2016.

Hong Sun, Nan Duan, Yajuan Duan, and Ming Zhou. Answer extraction from passage graph for question answering. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

Renxu Sun, Jing Jiang, Yee Fan, Tan Hang, Cui Tat-seng, and Chua Min-yen Kan. Using syntactic and semantic relation analysis in question answering. In *Proceedings of Fourteen Text Retrieval Conference (TREC-2005)*, 2005.

Stefanie Tellex, Boris Katz, Jimmy Lin, Aaron Fernandes, and Gregory Marton. Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of ACM Special Interest Group on Information Retrieval (SIGIR)*, 2003.

J. Turmo, P. Comas, S. Rosset, O. Galibert, N. Moreau, D. Mostefa, P. Rosso, and D. Buscaldi. Overview of QAst 2009. In *Proceedings of the CLEF 2009 Workshop*, 2009.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of NIPS*, 2017.

Ellen M. Voorhees and Dawn M. Tice. Building a question answering test collection. *ACM Special Interest Group on Information Retrieval (SIGIR)*, 2000.

Rui Wang and Günter Neumann. DFKI-LT at AVE 2007: Using recognizing textual entailment for answer validation. In *online proceedings of CLEF*, 2007.

Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.

Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of Association for Computational Linguistics (ACL)*, 2017.

Zhiguo Wang, Haitao Mi, and Abraham Ittycheriah. Sentence Similarity Learning by Lexical Decomposition and Composition. *arXiv preprint arXiv:1602.07019.*, 2016.

Dirk Weissenborn, Georg Wiese, and Laura Seiffe. Making neural qa as simple as possible but not simpler. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, 2017a.

Dirk Weissenborn, Georg Wiese, and Laura Seiffe. Fastqa: A simple and efficient neural architecture for question answering. *arXiv preprint arXiv:1703.04816*, 2017b.

Joseph Weizenbaum. ELIZA—a computer program for the study of natural language communication between man and machine. In *Communications of the Association for Computing Machinery (ACM)*, 1966.

Jason Weston, Sumit Chopra, and Bordes Antoine. Memory networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards AI complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2016.

Yuk Wah Wong and Raymond J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of Association for Computational Linguistics (ACL)*, 2007.

William Woods. Progress in Natural Language Understanding - An Application to Lunar Geology. In *Proceedings of the American Federation of Information Processing Societies (AFIPS)*, volume 42, 1973.

Min Wu, Mingyuan Duan, Samira Shaikh, Sharon Small, and Tomek Strzalkowski. ILQUA - an IE-driven question answering system. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC)*, 2005.

Pengtao Xie and Xing Eric. A Constituent-Centric Neural Architecture for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, 2017.

Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

Yi Yang, Wen tau Yih, and Christopher Meek. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1999.

Yi Yang, Wen-tau Yih, and Christopher Meek. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

X. Yao and B. Van Durme. Information extraction over structured data: Question Answering with Freebase. In *Proceedings of Association for Computational Linguistics (ACL)*, 2014.

Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. Answer extraction as sequence tagging with tree edit distance. In *Proceedings of HLT-The North American Chapter of the Association for Computational Linguistics (NAACL)*, 2013.

Wen-tau Yih, Ming-Wei Chang, Christopher Meek, and Andrzej Pastusiak. Question answering using enhanced lexical semantic models. In *Proceedings of Association for Computational Linguistics (ACL)*, 2013.

Wen-tau Yih, Xiaodong He, and Christopher Meek. Semantic parsing for single-relation question answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, page 643–648, 2014.

Wenpeng Yin, Sebastian Ebert, and Hinrich Schütze. Attention-based convolutional neural network for machine comprehension. *arXiv preprint arXiv:1602.04341v1*, 2015a.

Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *arXiv preprint arXiv:1512.05193*, 2015b.

Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. Deep learning for answer sentence selection. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS) - Deep learning workshop*, 2014.

J. M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic proramming. In *Proceedings of the National Conference on Artificial Intelligence*, 1996.

L. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Annual Conference in Uncertainty in Artificial Intelligence (UAI)*, 2005.

Junbei Zhang, Xiaodan Zhu, Qian Chen, Lirong Dai, and Hui Jiang. Exploring question understanding and adaptation in neural-network-based question answering. *arXiv preprint arXiv:1703.04617*, 2017.

Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

List of Figures

List of Tables

List of Abbreviations

ACC, Accuracy, Chapter 6

ATIS, Air Traffic Information System, Chapter 2

BOW, Bag Of Words, Chapter 1

BPTT, Back-Propagation Through Time, Chapter 1

CDNN, Constrained Deep Neural Network, Chapter 6

CNN, Convolution Neural Network, Chapter Introduction

COLING, The international conference on Computational Linguistics, Chapter 1

CoNLL, The conference on Computational Natural Language Learning, Chapter 1

DNN, Deep Neural Network, Chapter Introduction

EMNLP, The conference on Empirical Methods in Natural Language Processing, Chapter 1

GD, Gradient Descent, Chapter 1

GESD, the Geometric mean of Euclidean and Sigmoid Dot product, Chapter 6

GRU, Gated Recurrent Unit, Chapter 6

id, identifier, Chapter 6

ILP, Integer Linear Programming, Chapter 2

LR, Logistic Regression, Chapter 5

LSTM, Long Short-Term Memory, Chapter 1

MAP, Mean Average Precision, Chapter 1

MC, Machine Comprehension, Chapter 1

mid, machine identifier, Chapter 5

MQL, Meta-web Query Language, Chapter 1

MRR, Mean Reciprocal Rank, Chapter 1

NAACL, The North American Chapter of the Association for Computational Linguistics, Chapter 1

NLIDB, Natural Language Interface to Database, Chapter 2

NLP, Natural Language Processing, Chapter Introduction

NN, Neural Network, Chapter Introduction

PMI, Point-wise Mutual Information, Chapter 2

Q2AD, Quora Question Answering Dataset, Chapter Introduction

QA, Question Answering, Chapter Introduction

RNN, Recurrent Neural Network, Chapter Introduction

SGD, Stochastic Gradient Descent, Chapter 1

SP, Semantic Parsing, Chapter 2

SPARQL, SPARQL Protocol, and RDF Query Language, Chapter 1

SQ, SimpleQuestion, Chapter 3

SQuAD, Stanford Question Answering Dataset, Chapter 2

TSD, The international conference on Text, Speech, and Dialogue, Chapter 1

A. Structured Models

A.1 Features

The features in the Logistic Regression model are simply the words replaced with their word embeddings. (Please see Figure A.1). The word embeddings are obtained by either training word vectors on the dataset using Word2Vec toolkit (Mikolov et al., 2011) or using the pre-trained Word2Vec word vectors trained on GoogleNews. The Neural Network model uses the same features.

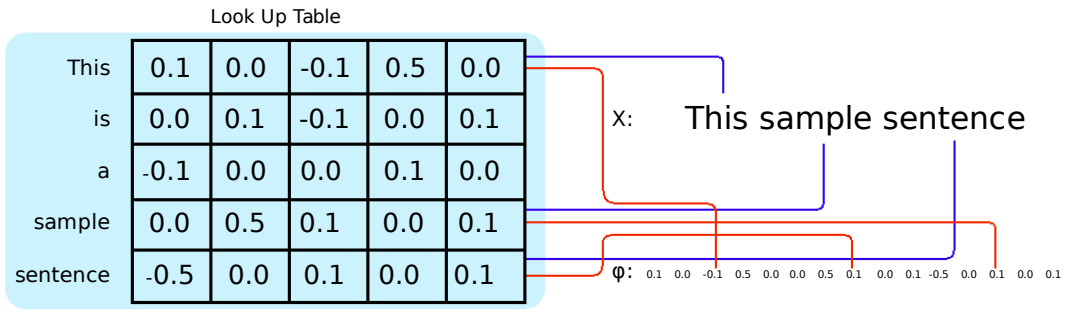


Figure A.1: The features in the Logistic Regression model

In the CNN model, two groups of features are used. First, the word tokens replaced with their embeddings like the way they were used in the models mentioned above. In the CNN the pre-trained word embeddings can be updated through training on the dataset as well. The second group of features is Part-Of-Speech tags of the words which are used in the second channel of the CNN model.

A.2 Hyper-parameters

In the Logistic Regression model, the hyperparameters are the dimensions of the word embeddings and the length of the questions[1]. In the Neural Network model in addition to question length and embedding dimension, the number of hidden

[1]To generate a fixed dimensional vector for all the questions, all of them are fixated into a predefined length.

layers and the number of neurons in each layer are hyper-parameters. The result of hyper-parameter tunning for the Logistic Regression and the Neural Network model are reported in Tables A.1 and A.2 respectively. The parameter question length is not changed in the NN and CNN models anymore since the tuned value for it in Logistic Regression experiment is used. In the following tables, all the scores are path accuracies in percent scale.

	Emb. Dimentions		Question Length			Emb. Training	
	100	300	10	20	30	trained on data	pretrained
Logistic Regression Model	59.5	61.2	48.9	61.2	59.7	61.2	60.9

Table A.1: Comparative results for different choices of hyper-parameters for the Logistic Regression model. The difference between using pre-trained embedding and training them on the dataset is not statistically significant.

	Emb. Dimensions		Num. of Hidden Layers		Num. of Neurons		
	100	300	1	2	512	1024	2048
Neural Network Model	61.7	63.89	63.89	52.8	60.7	63.89	57.4

Table A.2: Comparative results for different choices of hyper-parameters for the Neural Network model. The decrease in performance by an increase in complexity of the model like an increase in the number of layers of neurons is a signal of over-fitting.

Finally, in the CNN model, In addition to question length and embedding dimension, the number and the length of filters and the number of channels are hyper-parameters. The result of hyper-parameter tunning are reported in Table A.3.

	Emb. Dimensions		Num. of Channels		Num. of filters			Length of filters		
	100	300	1	2	128	256	512	2	3	4
CNN Model	63.41%	65.19	65.16	65.19	63.84	65.19	64.58	61.42	65.19	65.18

Table A.3: Comparative results for different choices of hyper-parameters for the CNN model. The numbers for length of filters represents the max length of the filters (i.e. 2 means filters with lengths 1 and 2, 3 means filters with lengths 1, 2, and 3, etc).

B. Sentence-Level Models

B.1 Hyper-parameters

In this model the dimension of the word embedding, the number of layers in the LSTMs, the drop-out rate, and the margin in the loss function are the hyper-parameters. The same model is used for Q2AD evaluation in the last chapter hence the same hyper-parameters applies there too. The results reported here are MRR on the TrecQA dataset. The decrease in the performance of the system with more LSTM layers is due to over-fitting. In the following tables, all scores are MRR in percent scale.

	Emb. Dimensions		LSTM Layers			drop-out ratio			Margin		
	100	300	64	128	256	0.3	0.5	0.7	0.02	0.05	0.08
Sentence-Level Model	88.2	89.5	84.7	89.5	87.1	87.4	89.5	81.7	88.4	89.5	87.9

Table B.1: Comparative results for different choices of hyper-parameters for the Sentence-level model

We also investigated the system performance on using various similarity measures, and the results are reported in Table B.2

Similarity measure	MRR(%)
Cosine	81.7
Polynomial	73.4
Radial Basis Function	84.2
Euclidean	79.6
Exponential	85.3
GESD	89.5

Table B.2: Evaluation on different similarity measures integrated in the model

C. Unstructured Models

C.1 Hyper-parameters

In addition to the type of constituents which were studied in the ablation study in Chapter 8, the dimension of word embeddings, the number of layers in the LSTMs, the drop-out rate, and the margin in the loss function are the hyper-parameters. In the following tables, all scores are F1 in percent scale.

	Emb. Dimensions		LSTM Layers			drop-out ratio			Learning Rate		
	100	300	64	128	256	0.1	0.3	0.5	0.0007	0.001	0.003
Bi-Attention Model	76.4	79.5	73.6	79.5	77.9	76.3	79.5	78.1	76.8	79.5	77.4

Table C.1: Comparative results for different choices of hyper-parameters for the unstructured model

The results of an ablation study on the effect of using various distance measures are reported in Table C.2

Distance measure	F1(%)
Cosine	72.3
Polynomial	75.8
Radial Basis Function	73.1
Euclidean	70.8
Exponential	74.9
GESD	79.5

Table C.2: The evaluation on different integrated distance measures in the model

C.2 Constituent Types

Other constituent types include: NP, ROOT, NNP, NN, JJ, VP, CD, PP, S, NNS, ADJP, SBAR, NP-TMP, QP, ADVP, VBG, DT, VBN, IN, NNPS, VB, RB, VBD, VBZ, JJR, VBP, UCP, X, CC, FRAG, WHNP, JJS, NAC, NX, FW, TO, RBR, PDT, PRN, INTJ, PRT, WHPP, PRP, SINV, WHADJP, MD, RRC, WHADVP

For a description of each type please refer to Manning et al. (2014)

D. Q2AD

Some samples of the questions in the Q2AD (Aghaebrahimian, 2017b) are listed in the following table.

Question	What's a good way to get stronger?
Sentence-level answers	You get stronger by giving your body a reason to get stronger. Liftings weights is one of the best ways to do this, but it is not the only way. You could, for example, use bodyweight exercises.
Word-level answers	giving your body a reason to get stronger Liftings weights bodyweight exercises
Question	As a conservative what bothers you about other conservatives?
Sentence-level answers	What bothers me most is when conservatives say things that aren't at all conservative. I also don't have a lot of respect for conservatives who seem only to care about money. I see a lot of hypocrisy amongst Christians who have become vicious in their hatred of Muslims, anger towards the government, and complete lack of virtue in both speech and practice.
Word-level answers	when conservatives say things that aren't at all conservative conservatives who seem only to care about money hatred of Muslims, anger towards the government, and complete lack of virtue in both speech and practice
Question	Why is Prince Philip not as popular as Queen Elizabeth II?
Sentence-level answers	Because he's not as courteous and gracious as she is. He's a joker, and some of his jokes are insulting or racist. He breaks the boundaries of formal protocol required by diplomacy, while she is the consummate diplomat.
Word-level answers	he's not as courteous and gracious as she is He's a joker He breaks the boundaries of formal protocol required by diplomacy.
Question	What effect would occur if I stopped eating everything but chicken nuggets?
Sentence-level answers	You'd become malnourished within a few days more than likely. You would also experience a great deal of constipation due to a lack of fiber as well as a myriad of differing illness (potentially including but not limited to scurvy, rickets, and weight gain) due to vitamin deficiency and the high fat content of deep fried nuggets. Eventually your body would begin to shut down due to lack of proper nutrition and you'd more than likely die.
Word-level answers	You'd become malnourished You would also experience a great deal of constipation Eventually your body would begin to shut down
Question	Why do so many people like Harry Potter and the Prisoner of Azkaban?
Sentence-level answers	My answer is: because Prisoner of Azkaban was the first book when the series showed its potential. Prisoner of Azkaban was the first book in the series that really impressed me. The third book was the one that changed my mind.
Word-level answers	Prisoner of Azkaban was the first book when the series showed its potential really impressed me changed my mind

Table D.1: The Q2AD sample questions and answers

E. List of Publications

- Ahmad Aghaebrahimian. Linguistically-Based Deep Unstructured Question Answering. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL 2018)*, Brussels, Belgium, 2018a
- Ahmad Aghaebrahimian. Deep Neural Networks at the Service of Multilingual Parallel Sentence Extraction. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING 2018)*, Santa Fe, New-Mexico, USA, 2018b
- Ahmad Aghaebrahimian. Constrained Deep Answer Sentence Selection. In *Proceedings of the 20th International Conference on Text, Speech, and Dialogue (TSD)*, 2017a
- Ahmad Aghaebrahimian. Quora Question Answer Dataset. In *20th International Conference on Text, Speech, and Dialogue (TSD)*, 2017b
- Ahmad Aghaebrahimian. Hybrid Deep Open-Domain Question Answering. In *Proceedings of the 8th Language and Technology Conference (LTC)*, 2017c
- Ahmad Aghaebrahimian and Filip Jurčíček. Open-domain Factoid Question Answering via Knowledge Graph Search. In *Proceedings of the Workshop on Human-Computer Question Answering, The North American Chapter of the Association for Computational Linguistics (NAACL)*, 2016a
- Ahmad Aghaebrahimian and Filip Jurčíček. Constraint-Based Open Domain Question Answering Using Knowledge Graph Search. In *Proceedings of the 19th International Conference on Text, Speech and Dialogue (TSD), LNAI 9924*, 2016b
- Ahmad Aghaebrahimian and Filip Jurčíček. Machine Learning for Semantic Parsing in Review. In *Proceedings of the 7th Langauge and Technology Conference (LTC), Poznan, Poland*, 2015a
- Ahmad Aghaebrahimian and Filip Jurčíček. Constraint-based Statistical Spoken Dialogue Systems. In *Proceedings of the young researcher roundtable in spoken dialog (YRRSDS)*, 2015b
- Ahmad Aghaebrahimian. Constraint-based Semantic Parsing. In *proceeding of the Week of Doctoral Studies, Charles University, Faculty of Mathematics and Physics*, 2015