Learning to Represent, Categorise and Rank in Community Question Answering

Daria Bogdanova

Diploma in Applied Mathematics and Computer Science

A dissertation submitted in fulfilment of the requirements for the award of

Doctor of Philosophy (Ph.D.)

to the

Dublin City University
School of Computing

Supervisors:
Dr. Jennifer Foster
Prof. Qun Liu

January 2018

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Ph.D. is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:

(Candidate) ID No.: 13211109

Date:

Contents

Abstract **xi**

Publications **xii**

Acknowledgements **xiii**

1 Introduction **1**

1.1 Question Types in Question Answering 4

1.2 Research Questions 9

1.3 Contributions 10

1.4 Thesis Structure 12

2 Deep Learning for Natural Language Processing **15**

2.1 Logistic Classifier 16

2.2 Artificial Neural Networks 18

2.2.1 Artificial Neuron 18

2.2.2 Activation Function 19

2.3 Architectures 19

2.3.1 Multilayer Perceptron 20

2.3.2 Convolutional Neural Networks 21

2.3.3 Recurrent Neural Networks 24

2.3.4 Other Architectures 34

2.4 Training Neural Networks 34

2.4.1 Parameter Initialisation 35

2.4.2 Stochastic Gradient Descent . . . 37
2.4.3 Overfitting and Regularisation . . . 37
2.5 Unsupervised Pretraining . . . 39
2.5.1 Word Embeddings . . . 40
2.6 Hyperparameter Tuning . . . 44
2.7 Summary . . . 47

3 Detecting Semantically Equivalent Questions in CQAs 48
3.1 Question Classification Task . . . 50
3.2 Methodology . . . 51
3.2.1 Neural Network Architecture . . . 52
3.3 Experimental Setup . . . 54
3.3.1 Datasets . . . 54
3.3.2 Baselines . . . 55
3.3.3 Word Embeddings . . . 56
3.3.4 Hyperparameters . . . 57
3.4 Comparison with Baselines . . . 57
3.5 Impact of Domain-Specific Word Embeddings . . . 58
3.6 Impact of Training Set Size . . . 60
3.7 Experiments on a Different Domain . . . 61
3.8 Error Analysis . . . 62
3.9 The Chapter Reexamined . . . 63
3.10 Related Work on Question Retrieval . . . 64
3.10.1 Rules and Templates for Question Retrieval . . . 64
3.10.2 Statistical Techniques for Question Retrieval . . . 65
3.10.3 Representation Learning for Question Retrieval . . . 67
3.11 Summary . . . 68

4 Learning to Rank Answers 69
4.1 Related Work . . . 70

4.1.1 Non-Factoid Answer Ranking . . . 70
4.1.2 Ranking Scenarios in CQAs . . . 71
4.1.3 Features for Non-Factoid Answer Ranking . . . 73
4.1.4 Feature-Based Predictors . . . 82
4.1.5 Beyond Feature-Based Approaches . . . 83
4.2 Methodology . . . 86
4.2.1 Learning to Rank Answers . . . 88
4.2.2 Answer Ranking with Multilayer Perceptron . . . 90
4.2.3 Data Representation . . . 91
4.3 Resources . . . 92
4.4 Experimental Setup . . . 94
4.4.1 Baselines . . . 94
4.4.2 Evaluation Metrics . . . 95
4.4.3 Data Preprocessing . . . 96
4.5 Summary . . . 96

5 Answer Ranking with Paragraph Vector 98
5.1 Paragraph Vector . . . 99
5.1.1 Distributed Memory Model . . . 99
5.1.2 Distributed Bag-of-Words . . . 101
5.2 Experiments . . . 102
5.2.1 Ask Ubuntu Question Representation . . . 104
5.2.2 DBOW versus DM . . . 105
5.2.3 The Impact of the Paragraph Vector Size . . . 107
5.2.4 Paragraph Vector Representations for New Documents . . . 108
5.2.5 The Impact of the Pretraining Corpus . . . 110
5.3 Summary . . . 111

6 Learning Representations for Answer Ranking 112
6.1 RNN Encoder for Answer Ranking . . . 113

6.1.1 Prediction and Training . . . 114
6.1.2 LSTM versus GRU for Answer Ranking . . . 116
6.1.3 Augmenting the Representations . . . 118
6.2 Answer Ranking with Convolutional Neural Networks . . . 121
6.2.1 Hyperparameters . . . 124
6.2.2 CNN versus RNN for Answer Ranking . . . 124
6.3 Multi-Channel Recurrent Convolutional Neural Network . . . 125
6.4 Summary . . . 127

7 Further Analysis of Answer Ranking 129
7.1 Character-level versus Word-level Embeddings . . . 130
7.2 Injecting Discourse Features into the Neural System . . . 131
7.2.1 Discourse Features . . . 132
7.3 The Impact of Pretrained Word Embeddings . . . 134
7.4 Analysis . . . 138
7.5 Experiments on SemEval Data . . . 141
7.6 Summary . . . 143

8 Conclusion 145
8.1 Future Work . . . 148
8.1.1 Creation of Gold Standards . . . 148
8.1.2 Developing Interpretable Neural Architectures . . . 149
8.1.3 Developing Strategies for Hyperparameter Tuning . . . 150
8.1.4 Question Answering Evaluation . . . 150
8.1.5 Question Type Classification . . . 151
8.1.6 End-to-End Live Question Answering . . . 153

Bibliography 155

List of Figures

1.1 Example of a troubleshooting query. 2

2.1 Example of a model for spam classification. 20
2.2 Illustration of a multilayer perceptron with one hidden layer. 21
2.3 Illustration of a two-dimensional convolution. 23
2.4 Illustration of a max-pooling operation with a kernel of size 2x2 and a stride of 1. 24
2.5 Illustration of a convolutional neural network applied to a sentence. . 25
2.6 Illustration of a vanilla recurrent neural network. 26
2.7 Illustration of an unrolled vanilla recurrent neural network. 26
2.8 Illustration of an LSTM cell. 28
2.9 Illustration of a GRU cell. 29
2.10 Illustration of a bidirectional recurrent neural network. 30
2.11 Illustration of a stacked bidirectional RNN with two layers. 31
2.12 Illustration of an RNN encoder-decoder architecture 33
2.13 Illustration of an MLP before and after dropout. 39
2.14 High-level illustration of a CBOW model. 40
2.15 High-level illustration of a skip-gram model. 40
2.16 Illustration of the CBOW model for the word *panda* in context *the panda eats*. 42
2.17 Illustration of the skip-gram model predicting the words *the* and *eats* given the word *panda*. 43

2.18 Illustration of the learning process with a high learning rate 45
2.19 Illustration of the learning process with a low learning rate 45

3.1 Convolutional neural network for semantically equivalent questions detection. 53
3.2 CNN accuracy depending on the size of word embeddings 60
3.3 Development accuracy for the baseline and the CNN depending on the size of training set. 61

4.1 Illustration of a Yahoo! Answers Thread. 70
4.2 Illustration of a typical deep learning architecture for answer ranking. 84
4.3 Flowchart of our approach to answer ranking. 90

5.1 Illustration of the DM model for the word *panda* in context *the panda eats*. 100
5.2 Illustration of the DBOW model for learning a vector representation of the sentence *The panda eats* 102
5.3 Illustration of the method based on the Paragraph Vector and an MLP. 104
5.4 Development P@1 and test P@1 for the DBOW model with 50, 100, 200, 300 and 400-dimensional representations on the YA dataset. . . . 107

6.1 RNN-MLP model for answer ranking. 115
6.2 Example of a factoid question from TREC QA dataset with its correct answer and a possible alignment between them. Note that attention mechanisms define a *soft* alignment rather than the precise alignment represented here. 119
6.3 RNN-MLP model with explicitly encoded interactions between question and answer. 120
6.4 Illustration of a CNN encoder for answer ranking. 123
6.5 Illustration of MC-RCNN model. 127

7.1 Illustration of a neural architecture that incorporates additional features. 132
7.2 Feature generation for the discourse marker model of Jansen et al. (2014) 133
7.3 Average P@1 of the LSTM-MLP-Discourse versus the Random baseline on the test questions from most common YA categories. 140

List of Tables

1.1 Comparison of factoid and non-factoid question answering. 6

3.1 An example of semantically equivalent questions from Ask Ubuntu community. 49

3.2 Development Accuracy and best parameters for the baselines and the Convolutional Neural Network. 59

3.3 CNN and SVM accuracy on the development and the test set using the full training set. 59

3.4 Development Accuracy of the CNN with word embeddings pretrained on different corpora. 60

3.5 Convolutional Neural Network Accuracy tested on Meta Stack Exchange community data. 62

4.1 Comparative summary of the feature-based approaches to non-factoid answer ranking. 80

4.2 Comparative Summary of Neural Approaches in CQA 87

4.3 Example question and answers from the Yahoo! Answers dataset. . . 93

4.4 Example question and answers from the Ask Ubuntu dataset. 93

4.5 Comparative statistics on the datasets used in the answer reranking experiments. 94

5.1 Details on the corpora used to train the Paragraph Vector models. . . 103

5.2 DBOW performance: *title* versus *body* representation of the AU data. 105

5.3 Answer ranking results of the Paragraph Vector model in combination with an MLP. . . . 106
5.4 Comparison of the MLP performance using the extracted PV representations versus using the inferred PV representations. . . . 108
5.5 Comparison of the MLP performance using the DBOW representations inferred using a model trained on in-domain data versus the one trained on Wikipedia. . . . 110

6.1 Number of trainable parameters of the RNN-MLP model. . . . 116
6.2 Performance of the GRU and the LSTM encoders versus the baselines for answer ranking. . . . 117
6.3 Comparison of variations of encodings with LSTMs for answer ranking. 121
6.4 Answer ranking performance when using the RNN versus the CNN encoder. . . . 124
6.5 Performance of the system with a MC-RCNN encoder versus the RNN and the CNN-based systems. . . . 128

7.1 Answer reranking performance of different models when using word-level versus character-level embeddings. . . . 131
7.2 Experimental results on the test set for different encoders with and without discourse features. . . . 135
7.3 Details on the corpora used to pretrain the skip-gram model. . . . 135
7.4 Performance of the best performing models with random and pre-trained embeddings. . . . 137
7.5 Example incorrect predictions of the system on the Yahoo! Answers dataset. . . . 139
7.6 Example incorrect predictions of the system on the Ask Ubuntu dataset. 140
7.7 Details about Semeval 2016 Task 3 Subtask A data. . . . 141
7.8 MAP on the Semeval 2016 Task 3 Subtask A development set. . . . 142

7.9 Performance on the Semeval 2016 Task 3 Subtask A test set. Calculated using the official scorer. 142

Learning to Represent, Categorise and Rank in Community Question Answering

Daria Bogdanova

Abstract

The task of Question Answering (QA) is arguably one of the oldest tasks in Natural Language Processing, attracting high levels of interest from both industry and academia. However, most research has focused on factoid questions, e.g. *Who is the president of Ireland?* In contrast, research on answering non-factoid questions, such as manner, reason, difference and opinion questions, has been rather piecemeal. This was largely due to the absence of available labelled data for the task. This is changing, however, with the growing popularity of Community Question Answering (CQA) websites, such as Quora, Yahoo! Answers and the Stack Exchange family of forums. These websites provide natural labelled data allowing us to apply machine learning techniques.

Most previous state-of-the-art approaches to the tasks of CQA-based question answering involved handcrafted features in combination with linear models. In this thesis we hypothesise that the use of handcrafted features can be avoided and the tasks can be approached with representation learning techniques, specifically deep learning.

In the first part of this thesis we give an overview of deep learning in natural language processing and empirically evaluate our hypothesis on the task of detecting semantically equivalent questions, i.e. predicting if two questions can be answered by the same answer.

In the second part of the thesis we address the task of answer ranking, i.e. determining how suitable an answer is for a given question. In order to determine the suitability of representation learning for the task of answer ranking, we provide a rigorous experimental evaluation of various neural architectures, based on feedforward, recurrent and convolutional neural networks, as well as their combinations.

This thesis shows that deep learning is a very suitable approach to CQA-based QA, achieving state-of-the-art results on the two tasks we addressed.

Publications

Large portions of Chapter 3 have appeared in the following paper:

> Bogdanova, D., dos Santos, C., Barbosa, L., and Zadrozny, B. (2015). Detecting semantically equivalent questions in online user forums. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 123–131, Beijing, China

Large portions of Chapter 5 have appeared in the following paper:

> Bogdanova, D. and Foster, J. (2016). This is how we do it: Answer reranking for open-domain how questions with paragraph vectors and minimal feature engineering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1290–1295, San Diego, California

Large portions of Chapters 6 and 7 have appeared in the following paper:

> Bogdanova, D., Foster, J., Dzendzik, D., and Liu, Q. (2017). If you can't beat them join them: Handcrafted features complement neural nets for non-factoid answer reranking. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 121–131, Valencia, Spain

The author of this thesis also coauthored a paper, the contents of which are not included in this thesis:

> dos Santos, C., Barbosa, L., Bogdanova, D., and Zadrozny, B. (2015). Learning hybrid representations to retrieve semantically equivalent questions. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 694–699, Beijing, China

and a patent (rights belong to International Business Machines Corporation):

> De Andrade Barbosa, L., Bogdanova, D., Kormaksson, M., dos Santos, C., and Zadrozny, B. (2017). Machine learning and training a computer-implemented neural network to retrieve semantically equivalent questions using hybrid in-memory representations. US Patent 9,659,248

Acknowledgements

First of all, I would like to thank Jennifer Foster for being the best supervisor I could have asked for, for all scientific and emotional support she gave me. A special huge thank you for reading countless drafts with tons of misplaced determiners and giving me very valuable and prompt feedback!

I would also like to thank my second supervisor Qun Liu for his valuable advice that helped me a lot especially when Jennifer was on maternity leave. A special thanks to Josef van Genabith who was supervising me in the beginning of this PhD, and whom I owe the exciting topic I've chosen to work on.

My internship at IBM Research Brazil in 2014 influenced a lot the path I've taken in my research. A big thank you to Cícero Nogueira dos Santos whom I was very lucky to have as a mentor.

I should also deeply thank another mentor I had, Majid Yazdani, who taught me a lot about deep learning during my internship at LinkedIn in 2016. I am also very thankful to Juan, David, Mary, Deirdre, Matthieu, Chirag and the rest of the team at LinkedIn for their support and for all I've learnt from them.

Another special thanks to Joan Pastor who shared with me many useful practical tips on training and tuning neural nets.

I would also like to thank my fellows Iacer, Peyman and Chris for creating a deep learning atmosphere in the lab, for many great discussions we had and for being ready to share the GPU nodes. Thanks to Joachim Wagner for doing a great job maintaining the Maia cluster where I ran most of my experiments. Thanks to Daria and Henry for the cake breaks in Helix, those really helped me through the writing. Thanks to David, Flor, Eva and Dimi for their support and a good company! A special thanks to Lena for being a great friend and for being always there when I needed to talk.

I am also deeply grateful to Boris Novikov, the first advisor I had back at St.Petersburg university. Without him, I would not be where I am.

I want to thank my mother and my aunt-grandmother for caring so much for me, and for having done their best to give me a good education. Finally, a big thank you to Ximo who survived living with me while I was working on my thesis, for the nights he could not sleep because I was having nightmares about dropout (see Section 2.4.3), for being so patient and kind to me all this time.

Chapter 1

Introduction

Searching for information online has become a part of our day-to-day life. Modern search engines usually deal very well with simple fact seeking queries, such as finding out when Saint Patrick's Day is celebrated or looking up a phone number of a nearby restaurant. However, searching may become much more tiresome when the search goes beyond look up or fact retrieval (Marchionini, 2006; Cartright et al., 2011). This is the case when one is looking for a solution to a complex problem. An example of such a problem is shown in Figure 1.1 – the search results do not show a straightforward solution for this troubleshooting query. Community question answering websites (CQA), such as Quora[1], Yahoo! Answers[2] and Stack Exchange[3], were designed to address this issue and allow users to obtain answers to their questions directly from other users. A CQA can be viewed as a particular type of web forum designed to facilitate finding answers to questions. Like traditional web forums, CQAs are often organised by topic, e.g. *programming* or *travelling*. However, unlike traditional web forums, in CQAs social conversational posts, e.g. *how are you today?*, are not allowed or at least are discouraged and penalised. Nonetheless, these forums do not restrict information seeking questions to any particular type and contain a large proportion of non-factoid and narrative questions that remain

[1] `http://www.quora.com`
[2] `http://www.answers.yahoo.com`
[3] `https://www.stackexchange.com/`

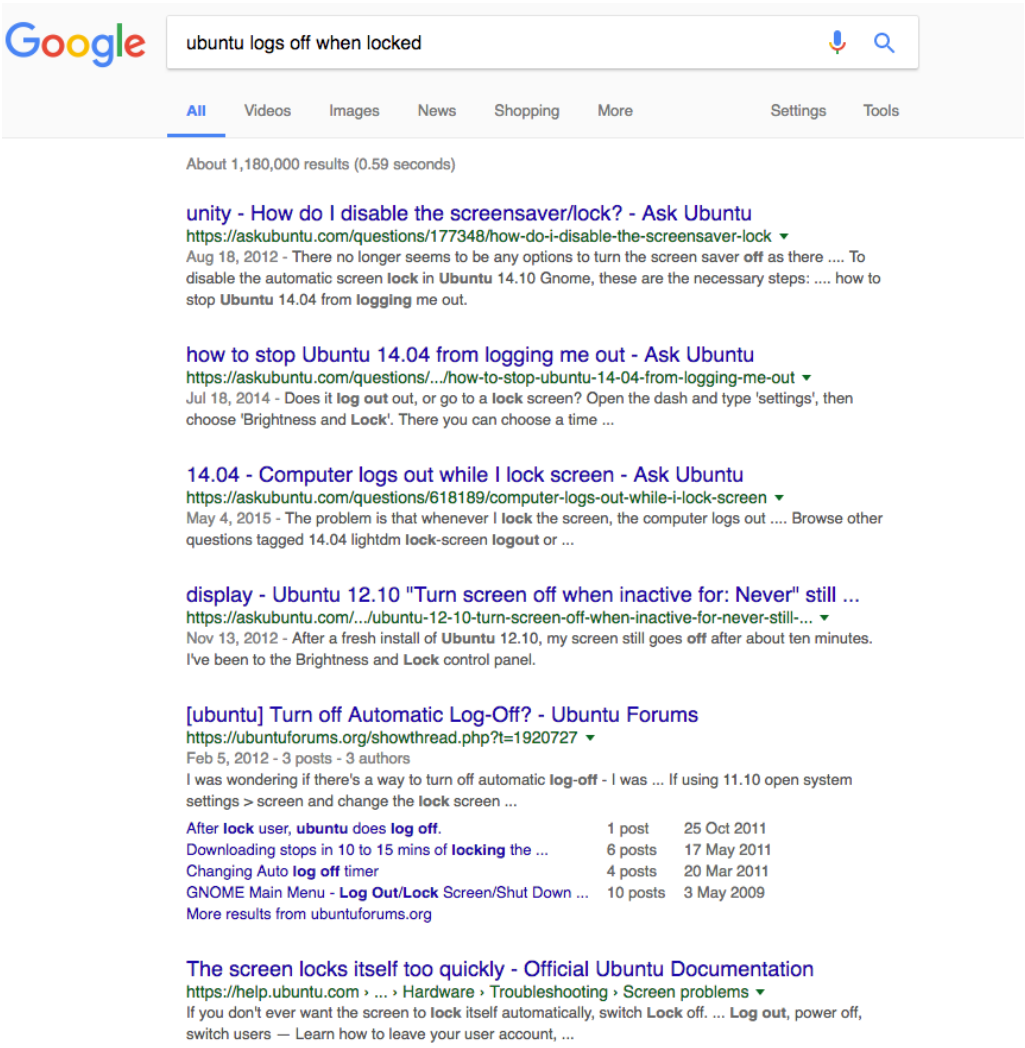

Figure 1.1: Example of a troubleshooting query, where Google search results do not readily provide the solution, with most search results coming from various user forums.

a challenge not only for search engines but also for modern question answering systems. Note that questions asked on CQAs are often not questions in the strict sense, as people use these forums to find solutions to the problems, e.g. *"I recently updated firewall setting (...) and now I am unable to download torrents (...)"*[4] Before the advent of CQAs, this kind of question has been rarely addressed in automatic question answering studies, mostly due to the absence of labelled data.

CQAs offer large amounts of questions along with their answers, allowing machine learning approaches to learn how to answer these questions. In most CQAs,

[4] `https://askubuntu.com/questions/919790/unable-to-download-torrent-in-transmission-after-updating-firewall`

a user who posts a question can then accept one of the answers to it, i.e. label it as correct. These user-provided labels allow us to use this data as training data for machine learning techniques. Nonetheless, using CQAs as a source of answers for automatic question answering raises a few challenges. The first one relates to the fact that there are many ways to ask the same question. Thus, finding useful questions becomes challenging. The second challenge relates to answer quality, i.e. not all answers posted on a CQA are useful.

In this thesis we explore the use of CQAs as an answer source in question answering, particularly focusing on *deep learning* techniques. By deep learning we mean machine learning approaches involving artificial neural networks. The research described in this thesis was started in 2013 when deep learning approaches were not common in Natural Language Processing (NLP). A typical approach to many tasks before the rise of deep learning in NLP was in developing features for the task and then using machine learning to optimise weights of these features (Manning, 2017; LeCun et al., 2015). The success of such an approach was mostly due to the clever feature design and successful numerical optimisation. The feature design or feature engineering usually required much human effort and domain knowledge. In contrast, deep learning approaches learn the representations needed for the task automatically. The word *deep* in deep learning refers to learning *various levels* of representation, starting from raw data and gradually learning more and more abstract representations. CQA sites contain noisy user-generated data which poses a challenge for many state-of-the-art NLP tools including part-of-speech taggers and parsers (Foster et al., 2011) and named entity recognisers (Ritter et al., 2011). One reason to focus on neural approaches is that they hold the promise of obviating the need for feature engineering, and in doing so allowing us to avoid propagating errors made by external NLP tools. Moreover, these methods have recently shown a lot of promise for other NLP applications. In this thesis we hypothesise that the use of handcrafted features can be avoided and the tasks can be approached with representation learning techniques, specifically deep learning.

The rest of this chapter is structured as follows: in Section 1.1 we place the CQA-based approaches in the context of question answering research and discuss question types in QA. In Section 1.2 we formulate the research questions we are addressing in this thesis. In Section 1.3 we summarise the contributions of this thesis. We give an overview of each of the following chapters in Section 1.4.

1.1 Question Types in Question Answering

The task of Question Answering (QA) is arguably one of the oldest tasks in Natural Language Processing (NLP), attracting high levels of interest from both industry and academia. The goal of this task is to provide an answer to a given question. In fact, question answering can be split into several related areas that all share this goal but differ in the types of questions they aim to answer:

Factoid Question Answering aims at answering factoid questions, i.e. questions that "can be answered with simple facts expressed in short text answers" (Jurafsky and Martin, 2014). These answers are usually expressed as numeric or named entities. Soricut and Brill (2004) refer to factoid questions as "questions for which a complete answer can be given in 50 bytes or less, which is roughly a few words". Factoid QA is a widely addressed task (Ferrucci et al., 2010; Berant et al., 2013; Iyyer et al., 2014; Voorhees and Tice, 1999), and what is usually referred to as *question answering*. The popularity of research on this type of question was partially due to the Text Retrieval Evaluation Conference (TREC) that introduced a question answering track in 1999 and since then has encouraged many research studies by providing a platform for evaluation and making labelled datasets available. The TREC QA track organisers took care to "select questions with straightforward, obvious answers"(Voorhees and Tice, 1999) to facilitate manual assessment, e.g. the TREC questions *What is the name of the managing director of Apricot Computer?* and *What was the monetary value of the Nobel Prize in 1989?* Methods for factoid question an-

swering can be divided into systems based on a knowledge base and IR-based systems. Knowledge base systems (Berant et al., 2013; Fader et al., 2013) build a semantic representation of the question that is then used to query a database, such as Freebase.[5] Knowledge bases used for question answering contain *(entity, property, entity)* triples called assertions, e.g. (`Dublin, CapitalOf, Ireland`). The QA systems usually apply semantic parsing to questions, i.e. convert them to a logical form that can be executed on a knowledge base. An example of such a logical form is Lambda Dependency-Based Compositional Semantics (λ-DCS) proposed by Liang (2013) and used by Berant et al. (2013). This model simplifies lambda calculus for the purposes of question answering. For instance, the logical form for a "writer born in Dublin" would be `Profession.Writer` $\sqcap$ `PlaceOfBirth.Dublin`, where $\sqcap$ stands for logical intersection.

IR-based systems (Monz, 2004; Paşca, 2003) typically apply the following steps to perform question answering:

1. **question processing** in order to detect the type of the answer (person, location, number etc.). The question hierarchy of Li and Roth (2006) is often used. This hierarchy contains six coarse classes (`ABBREVIATION`, `NUMERIC VALUE`, `ENTITY`, `HUMAN`, `LOCATION` and `DESCRIPTION`) and fifty fine-grained classes (e.g. entities: `animal`, `colour`, `food`; locations: `city`, `country` etc.);

2. **query formulation** from the question. This may involve query reformulation or expansion (Lin, 2007), e.g. removing the *wh*-word: *When was the telephone invented?* can be reformulated as *telephone was invented*. In contrast to the knowledge base approaches, the IR approaches do not convert the question into a logical form, and use natural language queries instead;

[5] `http://www.freebase.com`, was discontinued and merged with Wikidata (`http://www.wikidata.org/`) in 2015.

	Factoid	Non-factoid
Answer	Short, usually a named entity	Usually longer, not a named entity
Evaluation	Automatic, String matching	Manual, laborious
Main Resources	TREC benchmarks	CQAs
Example	Question: *Who invented the telescope?* Answer Type: Person Answer: *Hans Lippershey*	Question: *How does the telescope work?* Answer Type: Paragraph Answer: *It uses two mirrors to magnify incoming light and form an image for the eye or an instrument (...)*

Table 1.1: Comparison of factoid and non-factoid question answering.

3. **document and passage retrieval**. Documents are retrieved using a search engine, then they are broken down into passages;
4. **answer processing** that extracts the answers from these passages and ranks them.

An overview of IR-based techniques for factoid QA can be found in Kolomiyets and Moens (2011).

Non-Factoid Question Answering: This task aims to answer **non-factoid (NF)** questions, i.e. questions that are not factoid, such as, for instance, manner (*how*) and reason (*why*) questions. These questions are sometimes also referred to as **narrative** questions. Non-factoid questions usually require a more complex and longer answer than factoid questions. Table 1.1 provides a comparison of the tasks of factoid and non-factoid question answering.

Research on answering non-factoid questions has been rather piecemeal, largely due to the absence of available labelled data for the task. Moreover, the nature of non-factoid questions does not allow automatic evaluation methods to be used, and thus, requires laborious manual evaluation.

Methods for non-factoid QA can be roughly divided into (1) Web-based and (2) CQA-based. The first group of methods adapts the information retrieval paradigm for factoid QA. First, candidate passages are retrieved using an information retrieval method, such as BM25 (Robertson et al., 1994), and then the passages are reranked using more expensive techniques. For instance, web-based approaches are presented by Keikha et al. (2014) and Yang et al. (2016). Their findings show that the task is challenging and existing methods do not perform well on this task. Yang et al. (2016) also showed that the IR-based approach to factoid QA of Yu et al. (2014) does not perform well when applied to non-factoid QA, however, adding additional semantic features, such as vectors obtained with Explicit Semantic Analysis (Gabrilovich and Markovitch, 2009), improves the performance.

The CQA-based methods use CQA websites as a source of answers. In fact, in CQA websites, the *questions* are not interpreted in the strict sense, i.e. they go beyond the sentence level and rather describe a problem. They may ask none (e.g. *When I login, nothing happens. I am presented with my desktop wallpaper. No Dash, no Launcher, nothing.*[6]) or several questions (e.g. *I'm absolutely new to Linux. I would like to know how to install Ubuntu alongside the pre-installed Windows 8+ OS. Should I do it with Wubi, or through the Live USB/DVD? What steps do I need to take to correctly install Ubuntu?*[7]) In this thesis, we focus on the CQA-based methods.

Multi-modal Question Answering: Given an image or a video, the task is to answer questions about this image or video. Antol et al. (2015) present the task of visual question answering (VQA) and release a dataset created by crowdsourcing. They suggest using the very deep convolutional network of Simonyan and Zisserman (2014) to embed the image; a deep long short term

[6] `https://askubuntu.com/questions/17381/unity-doesnt-load-no-launcher-no-dash-appears`

[7] `https://askubuntu.com/questions/221835/installing-ubuntu-alongside-a-pre-installed-windows-with-uefi`

memory (LSTM) network to embed the question; and a multilayer perceptron (MLP) to combine the two embeddings. We discuss these architectures in detail in Chapter 2. Finally, a softmax layer is used to predict the answer. Zhang et al. (2016) observed that this model relied mostly on the textual rather than the visual information, e.g. answering *yes* to all questions asking *Do you see a ... ?* achieved 87% accuracy. In order to emphasise the image understanding part of the task, they balanced the VQA dataset: for most questions, they added another image where the answer was different (e.g. the answer to *What is the dog wearing?* was *collar* for one image and *life jacket* for another). An overview of methods used in visual question answering can be found in Wu et al. (2016).

Artificial Intelligence Tests: This area aims at developing methods capable of general reasoning and natural language understanding. One of the main tasks this area investigates is reading comprehension, see, for instance, the Facebook bAbI tasks (Weston et al., 2015a). The bAbI tasks are twenty synthetic tasks aimed to test general text understanding and reasoning. The dataset contains simulations of different characters moving between locations and interacting with each other and with objects. Each task aims at modelling a different reasoning skill, for instance, basic deduction: *Sheep are afraid of wolves. Cats are afraid of dogs. Mice are afraid of cats. Gertrude is a sheep. What is Gertrude afraid of?* and counting: *Daniel picked up the football. Daniel dropped the football. Daniel got the milk. Daniel took the apple. How many objects is Daniel holding?*. Memory networks (Sukhbaatar et al., 2015) have been shown to achieve good performance on these tasks.

Rajpurkar et al. (2016) created another dataset for reading comprehension called the Stanford Question Answering Dataset (SQuAD). This dataset contains more than 100K questions with their answers identified as passages in corresponding Wikipedia articles. Currently, the best performing system on

the SQuAD dataset is R-Net (Wang et al., 2017), a model combining gated recurrent neural networks, attention mechanisms and pointer networks.

1.2 Research Questions

In this thesis we explore the use of CQA sites for question answering. While the main focus on the thesis is the task of answer ranking in CQAs, we also explore the task of predicting semantically equivalent questions.

Most previous approaches to non-factoid question answering were based on handcrafted features (Verberne et al., 2011, 2007; Higashinaka and Isozaki, 2008; Jansen et al., 2014; Fried et al., 2015). These approaches require human feature engineering, and are often difficult to adapt to other domains and datasets, e.g. for CQA websites, the website-specific metainformation, such as the number of good questions and answers posted by the same user, is often used. In this thesis, we focus on deep learning approaches. The first research question concerns the limits of these methods in detecting semantically equivalent questions:

1. *Is it possible to predict semantically equivalent questions in community question answering websites using a deep learning system and relying on textual information only?*

The second question concerns the limits of deep learning methods for the task of answer ranking in CQA:

2. *Can we rank answers to questions in community question answering websites without relying on handcrafted features?*

The rest of the research questions concern the neural approaches to the task of answer ranking. In particular, we explore several neural architectures for the task of answer ranking including convolutional and recurrent neural networks. We formulate the third research question as follows:

3. *Which neural architectures are most suitable for encoding questions and answers in answer ranking?*

Traditional feature-based and neural approaches are often viewed as opposed to each other. Our fourth research question concerns the possibility of combining the two approaches:

4. *Can feature-based and neural approaches be successfully combined for the task of answer ranking? Do neural systems for answer ranking benefit from the inclusion of tried-and-tested features for this task?*

Since we focus on CQAs, that do not restrict questions to any particular type, we investigate which questions are the most challenging from the point of view of automatic answer ranking:

5. *What kinds of questions pose the greatest challenge for the automatic answer ranking systems?*

1.3 Contributions

The contributions of this thesis are:

1. **Experiments on detection of semantically equivalent questions.** We define semantically equivalent questions as questions that can be adequately answered by the exact same answer and investigate the use of convolutional neural networks for predicting such questions in CQA sites. We show that convolutional neural networks provide good performance on this task. We also show that they need less training data than the baseline methods, i.e. support vector machines. The neural system for detecting semantically equivalent questions we present was developed in 2014, and it was, to the best of our knowledge, the first attempt to apply deep learning methods to this task.

2. **Survey of the research on non-factoid question answering.** Related work on community question answering spans different areas and communities. We

review the literature and provide a detailed overview of existing approaches to non-factoid answer ranking. We divide the methods into two groups: (1) feature-based methods, i.e. ones that perform the ranking using handcrafted features; and (2) neural methods, that achieve the ranking due to the representation capacity of the neural architecture. We first provide an overview of the features used to rank answers to non-factoid questions. Then we describe the neural approaches to the task as well as some related tasks. To the best of our knowledge, this is the first comprehensive survey of non-factoid question answering that includes neural approaches to this task.

3. **Experiments on neural approaches for CQA.** We investigate the use of artificial neural networks, i.e. deep learning, for the tasks of community question answering. We conduct extensive answer ranking experiments in two very different CQAs. We compare the performance of various neural architectures including the Long Short Term Memory networks and convolutional neural networks, we also investigate the impact of pretrained word embeddings on the performance of the neural systems. Overall, we show that neural systems provide new state-of-the-art results on these tasks.

4. **Multi-Channel Convolutional Recurrent Neural Network.** We propose a novel architecture called Multi-Channel Convolutional Recurrent Neural Network (MC-RCNN) for encoding sentences and documents. This architecture combines the benefits of recurrent neural networks with gating mechanisms that capture long-term dependencies, and convolutional neural networks, that capture local features. We experimentally show that this architecture is suitable for encoding questions and answers for the task of answer ranking.

5. **Combining neural systems with discourse features for answer ranking.** We show that a neural system for answer ranking can be extended and improved by inclusion of tried-and-tested features such as discourse features. We incorporate the discourse features proposed by Jansen et al. (2014) into our

neural architecture. We show that despite being often viewed as opposed, the neural approach and handcrafted features can be complimentary and their joint use can improve the overall performance of answer ranking systems.

1.4 Thesis Structure

This thesis is structured as follows:

Chapter 2

In this chapter, we review the basics of artificial neural networks, i.e. *deep learning*. We mainly focus on the techniques necessary to understand the content of this thesis. In particular, we introduce multilayer perceptrons, i.e. feedforward fully connected neural networks, as well as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The latter includes Long Short Term Memory networks (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Networks (Cho et al., 2014b). We provide a very brief overview of a few other architectures, including RNN encoder-decoder with attention (Bahdanau et al., 2014). We then explain how neural networks are trained, and also talk about parameter initialisation and hyperparameter tuning. We also discuss unsupervised pretraining of word embeddings at the end of this chapter.

Chapter 3

In this chapter, we introduce the task of detecting semantically equivalent questions in community question answering websites. We approach this task by using a convolutional neural network to encode the questions to a fixed-length vector, and then compare the vectors using cosine similarity. We experimentally show that this approach provides good results. We also investigate the impact of the word embeddings on the performance of this method by varying their dimensionality and the corpus used for their training. We also

investigate the impact of the training set size on the performance of the convolutional approach versus the support vector machine baseline.

Chapter 4

In this chapter, we present the task of answer ranking in community question answering websites. We start by providing a literature review on the topic. We first describe the traditional feature-based approaches to answer ranking, and then review neural approaches to this and related tasks. We present our general approach to the task of answer ranking where we encode the question and the answer and use a multilayer perceptron to score the answer. We introduce the experimental setup and the datasets we use in our experiments.

Chapter 5

In this chapter, we investigate the use of the Paragraph Vector model (Le and Mikolov, 2014) for the task of answer ranking. We first train this model in an unsupervised fashion and use the obtained vectors to initialise the representations for the questions and the answers. We compare the performance of the two Paragraph Vector models, i.e. the Distributed Bag-of-Words and the Distributed Memory. We investigate the impact of the dimensionality of the representations and the nature of the pretraining corpus.

Chapter 6

In this chapter, instead of relying on general purpose representations like in Chapter 5, we focus on learning representations for questions and answers simultaneously with learning the actual task. In particular, we start by encoding the questions and the answers using recurrent neural networks. We compare the performance of the two most common variants of recurrent neural networks, Long Short Term Memory networks and Gated Recurrent Networks. We also investigate the use of attention mechanisms for encoding questions and answers. In addition, we compare recurrent neural networks and convolutional neural networks for encoding questions and answers. Finally, we

combine the two architectures and propose a novel neural architecture that we call a Multi-Channel Convolutional Recurrent Neural Network.

Chapter 7

In this chapter, we extend the experiments presented in Chapter 6. First, we investigate the use of character-level instead of word-level word embeddings. We then suggest enriching the neural system with tried-and-tested features. We choose to use the discourse features introduced by Jansen et al. (2014). Our experiments show that the neural approach benefits from the inclusion of these features. We also investigate the impact of unsupervised pretraining of the word embeddings, and provide error analysis. Finally, we test some of our neural approaches on the dataset of the SemEval shared task on community question answering.

Chapter 8

In this chapter, we summarise the findings of this thesis. We outline some of the questions remaining open and suggest directions for future work.

Chapter 2

Deep Learning for Natural Language Processing

Neural networks (also known as artificial neural networks or ANNs) have a long history dating back to 1943 when a neurophysiologist Warren McCulloch and a logician Walter Pits wrote an article hypothesising about how brain neurons might work (McCulloch and Pitts, 1943). The first system modelling an artificial neuron was introduced by Rosenblatt (1957) and was called *Perceptron*. It was only in 1986 when the backpropagation algorithm was proposed by Rumelhart et al. (1986), that it became possible to train multilayered neural networks. Since the late 80s, neural networks were believed to be theoretical models that were impossible to be trained in practice. This changed in 2006 when Hinton et al. (2006) showed that a deep neural network could be effectively trained. Consequently, neural networks have regained their popularity in recent years, this time under the new name of *deep learning*, suggesting that these models are able to learn *multiple levels of composition* (Goodfellow et al., 2016).

Many fields, including Natural Language Processing, have seen their subareas moving towards deep learning approaches (Collobert et al., 2011; Goldberg, 2015). Deep learning has become popular in NLP because it obviates the need for feature engineering. Another reason for its success are the techniques for very efficient

unsupervised learning of word representations (Mikolov et al., 2013b) also known as word embeddings, that have been shown to be much more efficient than the count-based representations traditionally used in NLP (Baroni et al., 2014). In this chapter we will review the basics of neural networks, paying attention only to the models necessary to understand this thesis. For a review of greater breadth and depth we suggest to refer to Goodfellow et al. (2016).

Many NLP approaches involve (1) feature engineering, i.e. representing data as vectors of handcrafted features, and (2) model learning and/or inference: using a predictor on the obtained feature vectors. When dealing with the task of supervised classification, predictors such as logistic regression or a Support Vector Machines (SVM) classifier (Cortes and Vapnik, 1995) are often used. Neural networks have greater representation capacity, i.e. are much more powerful function approximators, than such linear predictors. This representational power makes it possible to avoid the first step of feature engineering and learn from raw features, such as words and characters, instead of feeding handcrafted features to a predictor.

This chapter is structured as follows: in Section 2.1 we describe a logistic classifier and how it is trained. In Section 2.2 we explain the concepts of an artificial neuron, an activation function and an artificial neural network. Section 2.3 describes several neural architectures including convolutional and recurrent neural networks. In Section 2.4 we explain how artificial neural networks are trained. Section 2.5 talks about unsupervised pretraining and word embeddings. In Section 2.6 we talk about the hyperparameters of neural systems. Finally, we summarise the chapter in Section 2.7

2.1 Logistic Classifier

In this chapter, we will focus on the task of classification, and before moving on to neural networks, we would like to demonstrate how a simple linear classifier, such as a **logistic regression** works. Classification solves the task of assigning a **label** to

an unseen input. The label is also called a **category** or a **class** – in this thesis we will use these terms interchangeably. Some examples of the classification task are: given a photo, decide if it is a dog, a cat or a capybara; given an image, decide if there is a person in it or not; given a text predict if it is a news article, a detective story or a poem. The task is called **binary** classification if there are only two classes to choose from, and **multiclass** classification if there are more than two possible classes.

Classification is typically a task of **supervised** learning: supervised means that we have a set of examples for which we know the correct label, and the goal is to use this data to build a classifier able to predict a label for an unseen example. Let's assume that the task is to predict one of three possible labels. We will use **one-hot encoding** for the labels: instead of representing them as one of the scalars *1*, *2* or *3*, each label will be a 3-dimensional vector with zeros everywhere but the position of the correct class. For example, *(0, 1, 0)* represents the label *2*.

We will denote the input to the classifier as $\mathbf{x}$ and the output as $\mathbf{y}$. The softmax classifier (also called logistic classifier and logistic regression[1], usually in the case of binary classification and when the logistic function is used instead of the softmax) applies a linear function to the inputs, generates the output and uses a softmax function to convert the output to class probabilities:

$$\text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{y} \tag{2.1}$$

The softmax function *squashes* a vector of arbitrary values into a vector of probabilities, i.e. all the values of the resulting vector are in the range of $(0, 1)$ and sum up to 1:

$$\text{softmax}(\mathbf{z}) = \frac{e^{\mathbf{z}}}{\sum_i e^{z_i}}$$

The matrix $\mathbf{W}$ and the vector $\mathbf{b}$ in Equation 2.1 are parameters to be learned on the training set. In order to learn these parameters we need to define a **loss**

[1]even though it has regression in its name, logistic regression is usually used as a classifier.

function (sometimes also called an **objective** or a **cost function**), that measures how different the predictions are from the true labels. The loss represents the error and should be high when the classifier performs poorly and low when it is doing well. Ideally, it should be equal to zero if and only if all predictions are correct. The loss function of the softmax classifier is usually the **cross-entropy**:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_i y_i \log(\hat{y}_i) \tag{2.2}$$

where $\hat{\mathbf{y}}$ is the predicted label and $\mathbf{y}$ is the true label. Cross-entropy is also known as negative log-likelihood. We will later use this loss for more sophisticated models. In order to train the classifier, we calculate the loss over the training set T and minimise it as a function of weights and biases:

$$\sum_{\mathbf{x},\mathbf{y}\in T} L(\text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b}), \mathbf{y}) \to \min_{\mathbf{W},\mathbf{b}}$$

This is usually done using gradient-based optimisation methods. As the gradient represents the slope of the surface created by the function, following the direction opposite to the gradient, we can reach a local minimum. This method is called **gradient descent** (Cauchy, 1847) – in practice, we usually use its variations such as stochastic gradient descent that will be discussed in Section 2.4.2.

2.2 Artificial Neural Networks

2.2.1 Artificial Neuron

As we have already mentioned above, the initial motivation behind neural networks was in imitating the human brain – hence, the name neural networks. In biology, a neuron is a nerve cell that is able to process and transmit information to other neurons. In the area of artificial intelligence, **neurons**, or **artificial neurons**, are mathematical functions that receive one or more inputs and produce an output. The simplest form of a neuron is a linear function, that receives the inputs x_i, calculates

a weighted sum of the inputs and adds biases:

$$y = \sum_i w_i x_i + b$$

2.2.2 Activation Function

Usually, the neurons are non-linear, as combining linear functions always results in a linear function. A non-linear function g, called an **activation function**, is applied to the output:

$$y = g(\sum_i w_i x_i + b)$$

The most common activation functions are:

- sigmoid function: $\sigma(x) = \frac{1}{1+e^{-z}}$
- rectified linear unit (ReLU): $\text{ReLU}(x) = max(x, 0)$
- hyperbolic tangent: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

A logistic classifier can be seen as one artificial neuron with a sigmoid activation. Neural networks get their name from the fact that they typically consist of many neurons. These neurons can be organised in layers, where the input to the neurons of the current layer are the outputs of the neurons of the previous layer. Figure 2.1 illustrates a deep model with several layers. The first layer receives the input in the form of characters, and each consecutive layer represents a more abstract representation of the input. Networks with several layers are referred to as *deep* networks.

2.3 Architectures

In this section we will briefly review the main NN architectures that we use in this thesis.

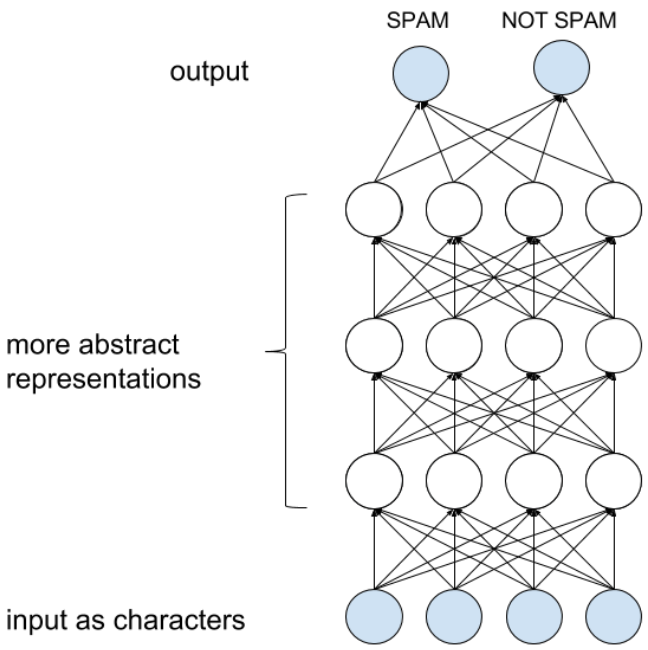

Figure 2.1: Example of a model for spam classification. The model receives the characters as input, and each layer provides a more abstract representation of the input. The final layer predicts one of the two classes.

2.3.1 Multilayer Perceptron

The most basic neural network architecture is a feedforward fully-connected neural network, also known as a **multilayer perceptron** (MLP). This architecture is called feedforward, as the information flows *forward* through the network, and fully-connected, as the neurons of each layer are connected to all the neurons of the previous layer. Figure 2.2 illustrates a feedforward neural network with one hidden layer. The inputs x are received by the network and are called the **input layer**, and the outputs y are produced by the network and are called the **output layer**. The layers *hidden* in between the input and the output layers (in this case the layer h) are called **hidden** layers. Each of the hidden units is a non-linear activation of a linear combination of the output of the previous layer, in this case the inputs x, i.e. $h_j = f(\sum_i w^0_{ij} x_i + b^0_j)$, and the output values y are activated linear combinations of

the hidden units: $y_j = f(\sum_i w_{ij}^1 x_i + b_j^1)$, where w_{ij}^k and b_j^k are parameters that are learnt during training.

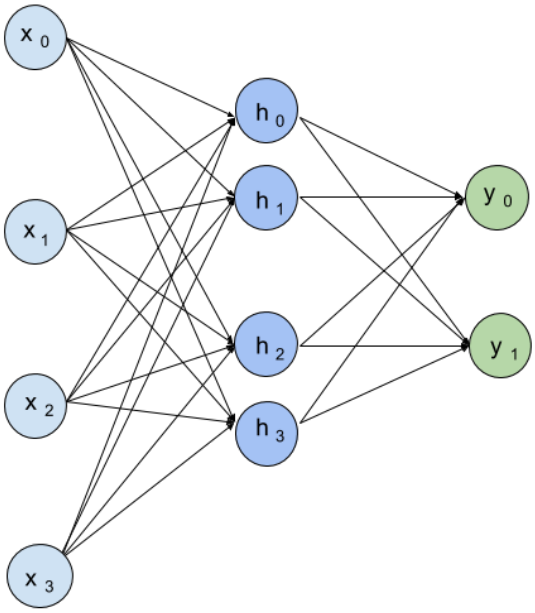

Figure 2.2: Illustration of a feedforward fully connected neural network, i.e. a multilayer perceptron, with one hidden layer.

2.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are designed to work with grids such as raw representations of images and recognise visual objects. They were inspired by the way the visual cortex works: the neurons respond to the stimuli of a restricted region called the receptive field, and the receptive fields of different neurons overlap. Similarly, in the CNN architecture the neurons are only connected to certain regions of the input or the previous layer, and these regions usually overlap. They are similar to the MLPs discussed in the previous section in their feedforward way of processing the information. However, there are a number of differences between these architectures. The MLPs treat every input independently, and all the neurons are fully connected to the neurons of the previous layer. A CNN could be seen as applying an MLP to a certain patch or kernel of the data, which is then shifted many times in order to cover the whole input. The weights of these MLPs are shared.

This operation is called **convolution**. This is illustrated in Figure 2.3. The input is divided into patches called **kernels** or **filters**, in fact, the window containing the patch is shifted each time, the shift of the kernel is called a **stride**. Each kernel is passed to a hidden layer that produces k outputs called **feature maps**. Using this convolution operation the input of size $height_{inp} \times width_{inp} \times p$ where p is the number of **channels** (e.g. 3 for red, green and blue) is mapped to an output of size $height_{out} \times width_{out} \times k$.

This convolution is usually combined with a **pooling** operation. The pooling combines vectors in a certain neighbourhood into a single vector by, for instance, summing them or getting their maximum or average. A common type of pooling is **max-pooling**, which is illustrated in Figure 2.4.

CNNs are designed to work on grid-shaped data, such as images. They are known to work very well on the task of handwritten digit recognition and achieve state-of-the-art performance on the MNIST[2] database of handwritten digits.[3] Most architectures include combinations of several convolutional and pooling layers. One of the first successful convolutional architectures was LeNet proposed by LeCun et al. (1998), which was a combination of a few convolutional and max-pooling layers with fully-connected layers. Another very famous convolutional architecture is AlexNet (Krizhevsky et al., 2012) that was initially developed for the task of image classification (Deng et al., 2009), but this model and its variations were also successfully applied to the tasks of object detection (Girshick et al., 2014), video classification (Karpathy et al., 2014), visual tracking (Wang and Yeung, 2013) and other computer vision tasks.

Even though the CNNs were designed to work with visual data, they have also been applied to textual data (Kim, 2014; Kalchbrenner et al., 2014; dos Santos and Gatti, 2014). The motivation for the use of CNNs for text is in their ability to convert the variable-sized input into a fixed-sized output, which is often needed in

[2]stands for Mixed National Institute of Standards and Technology, but usually known as just MNIST.

[3]`http://yann.lecun.com/exdb/mnist/`

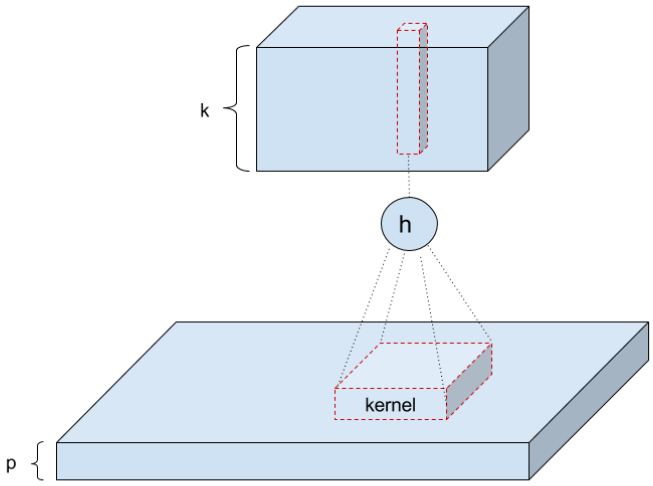

Figure 2.3: Illustration of a two-dimensional convolution. The input is divided into patches called kernels or filters. The window containing the patch is shifted each time, the shift of the kernel is called a stride. Each kernel is passed to a hidden layer that produces outputs called features maps.

NLP tasks. A typical CNN applied to texts is usually slightly different from the one applied to images, as images are usually represented as 3-dimensional grids, the first two dimensions are the spatial position and the third dimension corresponds to the colour channel. When dealing with textual data, the text is represented as word indices and then these indices are replaced with corresponding word embeddings. The convolution is then applied to word embeddings. Figure 2.5 illustrates a CNN that encodes a variable length text as a fixed-size vector. First, the words are represented as word embeddings: $\mathbf{e}_{w^{(1)}}, \mathbf{e}_{w^{(2)}}, \mathbf{e}_{w^{(3)}}, ..., \mathbf{e}_{w^{(n)}}$. These could be either initialised randomly or pretrained. For each word all the word vectors in the window of size k around it are concatenated,[4] and weights, biases and an activation are applied to the resulting vector:

$$\mathbf{z}^{(i)} = \sigma(\mathbf{W}[\mathbf{e}_{w^{(i-k)}}, ..., \mathbf{e}_{w^{(i)}}, ..., \mathbf{e}_{w^{(i+k)}}] + \mathbf{b})$$

[4]If there are not enough words on the left or on the right, a special padding vector is used.

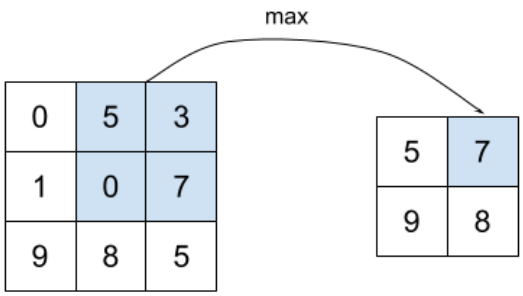

Figure 2.4: Illustration of a max-pooling operation with a kernel of size 2x2 and a stride of 1.

In Figure 2.5 k is equal to 1, i.e. only one word to the left and one word to the right are used to calculate the local features. The same weights $\mathbf{W}$ and biases $\mathbf{b}$ are used for all words, i.e. the weights and biases are shared for all the words, unlike the MLP architecture.

Finally, a max-pooling is applied to the output of the convolutional layer: each dimension i of the resulting representation r is maximum among the values of the vectors z along this dimension:

$$r_i = max(z_i^{(1)}, z_i^{(2)}, ..., z_i^{(n)})$$

We use a similar architecture in Chapter 3 for detecting semantically equivalent questions. A similar convolutional architecture but over character-level representation was used by dos Santos and Gatti (2014) for sentiment analysis.

2.3.3 Recurrent Neural Networks

The feedforward neural networks discussed in the previous sections have no cycles in them, and the information flows only forward, without any **feedback**. The family of neural networks with feedback connections are called recurrent neural networks (RNN). They were designed to work with sequences, as they allow the information to be *carried on through time*. This makes them very convenient to use on natural language data. When we speak and write, the meaning of each word is built on the meaning of whatever was said previously. For instance, the meaning of *bank*

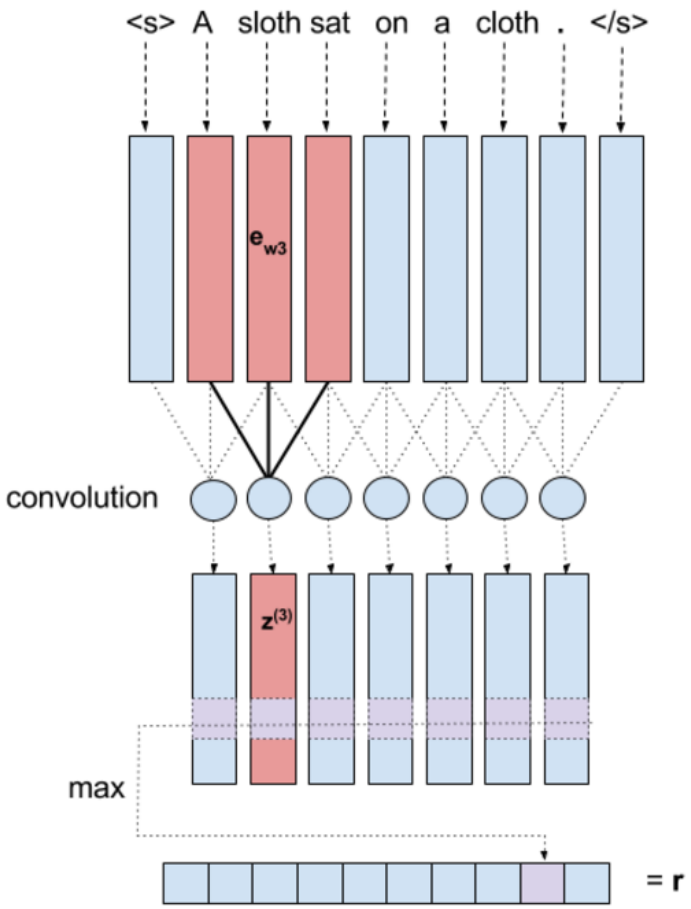

Figure 2.5: Example of a convolutional neural network applied to a sentence. The input words were transformed into the corresponding word embeddings. Then a convolution is applied to the embedding matrix. Each window of three words is convolved in one vector. Then a maximum per each dimension is taken.

if it is following the word *commercial* would be very different from its meaning if it has *river* just before it. This is what makes the RNNs so popular in the area of NLP. They have been successfully applied to a variety of NLP tasks including language modelling (Mikolov et al., 2010, 2011), natural language generation (Wen et al., 2015; Sutskever et al., 2011), machine translation (Auli et al., 2013; Bahdanau et al., 2014) and sentiment analysis (Tang et al., 2015a).

Figure 2.6 illustrates the way a simple recurrent network is usually visualised. It has a cycle: every new input to this network is first concatenated with the output this same network produced before and then passed to the network. This is also sometimes visualised as many copies of the same network, or the same network ap-

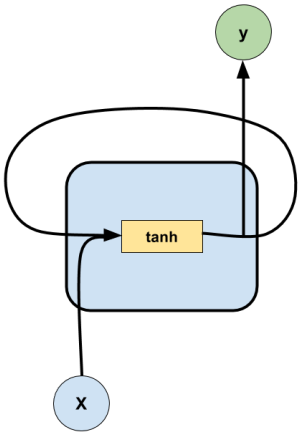

Figure 2.6: Vanilla RNN.

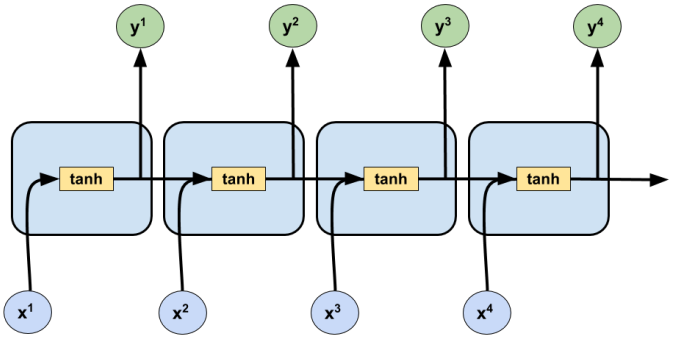

Figure 2.7: Unrolled vanilla RNN.

plied at several steps. It is important that these copies share the same weights, and are rather instances of the same network than its copies. Figure 2.7 shows a recurrent network **unrolled** in time, i.e. recurrent network represented as a multilayer network, where the number of layers is unlimited.

In the simplest version of the RNN, that is usually known as a *vanilla RNN* or a *tanh RNN*, the input sequence $\mathbf{x}$ is passed to the single hidden layer, and the hidden state $\mathbf{h}$ and the output $\mathbf{y}$ at each step are calculated as follows:

$$\mathbf{h}^{(t)} = \tanh(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}) \tag{2.3}$$

$$\mathbf{y} = \text{softmax}(\mathbf{V}\mathbf{h}^{(t)} + \mathbf{c}) \tag{2.4}$$

where $\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{b}$ and $\mathbf{c}$ are the network's parameters tuned during training.

In practice, the vanilla RNNs struggle with representing long-term dependencies due to the vanishing gradient problem. Imagine an unrolled vanilla RNN, which is essentially a very deep multilayer perceptron that shares weights. When we apply backpropagation to this network, we multiply the gradients many times, and they tend to either become very big or very small. The first problem is referred to as the **exploding gradient** problem, and is dealt with by **gradient clipping** (Pascanu et al., 2013), i.e. not allowing the gradients to grow higher than a certain threshold.

The second issue, i.e. the gradient becoming too small, is referred to as the **vanishing gradient** problem. Pascanu et al. (2013) suggest a regularisation technique that allows that problem to be avoided when training RNNs. Another solution to deal with the vanishing gradient problem is the gating mechanism implemented in the form of Long Short Term Memory networks described in the next section.

2.3.3.1 Long Short Term Memory Networks

Long short term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) are a type of recurrent neural network that were designed to cope well with the vanishing gradient problem via a **gating** mechanism. Central to LSTMs is the concept of an internal **state** that stores the information and serves as the *memory*[5] of the network. Let's denote this state at step t as $\mathbf{c}^t$. The gates control what information should be added to that memory, removed from there or used to generate the output at each step.

A gate is usually a sigmoid layer, that controls how much information can pass through this gate. It can be seen as a parametrised probability of passing through it. The LSTM defines three gates, the input gate $\mathbf{i}_t$, the forget gate $\mathbf{f}_t$ and the output gate $\mathbf{o}_t$:

$$\mathbf{f}^{(t)} = \sigma(\mathbf{W}_f \mathbf{x}^{(t)} + \mathbf{U}_f \mathbf{h}^{(t-1)} + \mathbf{b}_f) \tag{2.5}$$

$$\mathbf{i}^{(t)} = \sigma(\mathbf{W}_i \mathbf{x}^{(t)} + \mathbf{U}_i \mathbf{h}^{(t-1)} + \mathbf{b}_i) \tag{2.6}$$

$$\mathbf{o}^{(t)} = \sigma(\mathbf{W}_o \mathbf{x}^{(t)} + \mathbf{U}_o \mathbf{h}^{(t-1)} + \mathbf{b}_o) \tag{2.7}$$

These gates control what is to be added to the state $\mathbf{c}^t$, what is to be forgotten and what is to be outputted at each step. First, the network generates a new candidate state $\widetilde{\mathbf{c}}^{(t)}$:

$$\widetilde{\mathbf{c}}^{(t)} = \tanh(\mathbf{W}_c \mathbf{x}^{(t)} + \mathbf{U}_c \mathbf{h}^{(t-1)} + \mathbf{b}_c) \tag{2.8}$$

[5]Note, this vector does not refer to the same concept of *memory* as in memory networks and Neural Turing Machines, where *memory* states for the external memory with read and/or write access.

How much of the current state should be forgotten and replaced by the candidate state is decided using the forget and the input gates:

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \star \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \star \widetilde{\mathbf{c}}^{(t)} \tag{2.9}$$

Finally, how much information from the state should be outputted is decided using the output gate:

$$\mathbf{h}^{(t)} = \boldsymbol{o}_t \star \tanh(\mathbf{c}^{(t)}) \tag{2.10}$$

An illustration of the LSTM cell is shown in Figure 2.8. A detailed and very clear explanation of the LSTM is given in the blog of Christopher Olah.[6]

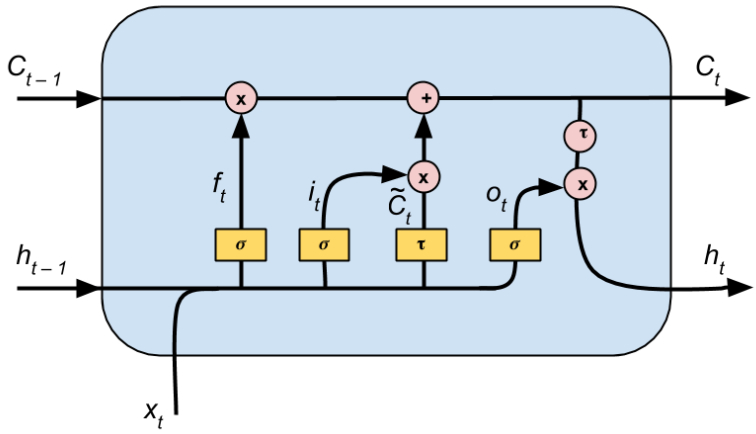

Figure 2.8: Illustration of an RNN with LSTM cell. The red circles stand for element-wise operations, the yellow squares denote non-linear layers. τ stands for hyberbolic tangent.

2.3.3.2 Gated Recurrent Neural Networks

The Gated Recurrent Unit (GRU) (Cho et al., 2014b) is a more recent variation of the LSTM. It uses two gates instead of three, the **update** gate:

$$\mathbf{z}^{(t)} = \sigma(\mathbf{W}_z \mathbf{x}^{(t)} + \mathbf{U}_z \mathbf{h}^{(t-1)} + \mathbf{b}_z) \tag{2.11}$$

[6] `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

and the **reset** gate:

$$\mathbf{r}^{(t)} = \sigma(\mathbf{W}_r \mathbf{x}^{(t)} + \mathbf{U}_r \mathbf{h}^{(t-1)} + \mathbf{b}_r) \tag{2.12}$$

It also has only one state $\mathbf{h}^{(t)}$ instead of having separate cell states and hidden states, as the LSTM does. The candidate state at each step is computed as follows:

$$\widetilde{\mathbf{h}}^{(t)} = \tanh(\mathbf{W}\mathbf{x}^{(t)} + \mathbf{U}(\mathbf{r}^{(t)} \star \mathbf{h}^{(t-1)}) + \mathbf{b}) \tag{2.13}$$

and the state is a linear interpolation between the previous state and the current candidate state:

$$\mathbf{h}^{(t)} = (\mathbb{1} - \mathbf{z}^{(t)}) \star \mathbf{h}^{(t-1)} + \mathbf{z}^{(t)} \star \tilde{\mathbf{h}}^{(t-1)} \tag{2.14}$$

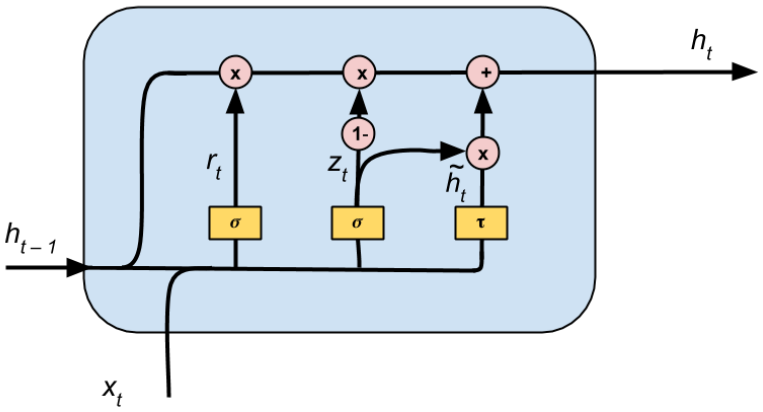

Figure 2.9: Illustration of an RNN with a GRU cell. The red circles stand for element-wise operations, the yellow squares denote non-linear layers. τ stands for hyberbolic tangent.

An illustration of a GRU cell is shown in Figure 2.9. Chung et al. (2014) evaluate GRUs versus LSTMs on the tasks of music and speech signal modelling and show that the two achieve similar performance, even though the GRU has fewer trainable parameters, and both LSTM and GRU are much more powerful than the vanilla RNNs. RNNs with GRUs called Gated Recurrent Networks are widely used in the area of neural machine translation (Firat et al., 2016; Chung et al., 2016).

There are many other variations of the RNN cell, including the peephole LSTM cell (Gers and Schmidhuber, 2000), which uses the concatenation of the previous state, previous output and the current input, in contrast to traditional LSTM that concatenates only the latter two.

2.3.3.3 Bidirectional and Stacked RNNs

The RNNs we described in the previous sections can be run from left to right or right to left. Recent inputs to an RNN have a stronger impact on its final output. This means that if we deal with natural language text in a left-to-right language like English, we pay more attention to the end of the sentence, which is not always desirable. A **bidirectional RNN** (Schuster and Paliwal, 1997) is designed to overcome these issues.

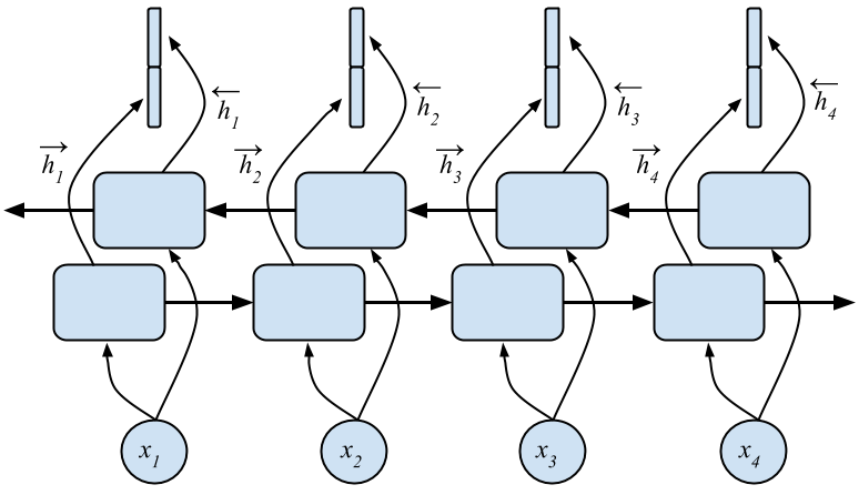

Figure 2.10: Illustration of a bidirectional recurrent neural network.

A bidirectional RNN consists of two separate unidirectional RNNs, that are run on the input in opposite directions, i.e. the **forward** and the **backward** RNN. The outputs of the two RNNs are merged, i.e. concatenated at each step. This is illustrated in Figure 2.10. Let's denote the outputs of the forward RNN (running from left to right) as $(\overrightarrow{\boldsymbol{h_1}}, \overrightarrow{\boldsymbol{h_2}}, \overrightarrow{\boldsymbol{h_3}}, ..., \overrightarrow{\boldsymbol{h_n}})$, and the outputs of the backward RNN as

$(\overleftarrow{\boldsymbol{h_1}}, \overleftarrow{\boldsymbol{h_2}}, \overleftarrow{\boldsymbol{h_3}}, ..., \overleftarrow{\boldsymbol{h_n}})$. Then, the output of the biRNN is

$$[\overleftarrow{\boldsymbol{h_1}}, \overrightarrow{\boldsymbol{h_1}}], [\overleftarrow{\boldsymbol{h_2}}, \overrightarrow{\boldsymbol{h_2}}], [\overleftarrow{\boldsymbol{h_3}}, \overrightarrow{\boldsymbol{h_3}}], ..., [\overleftarrow{\boldsymbol{h_n}}, \overrightarrow{\boldsymbol{h_n}}]$$

This applies to an RNN with any type of cell, e.g. LSTM. In this case it is referred to as bidirectional LSTM.

Another common augmentation of the RNN architecture is a **stacked RNN** or a **deep RNN** (Graves et al., 2013). It also combines several recurrent networks, but in contrast to the bidirectional RNN, where the two networks are independent, in this architecture one RNN receives the outputs of the other RNN as inputs. This could be done with either uni- or bidirectional networks. We illustrate this in Figure 2.11.

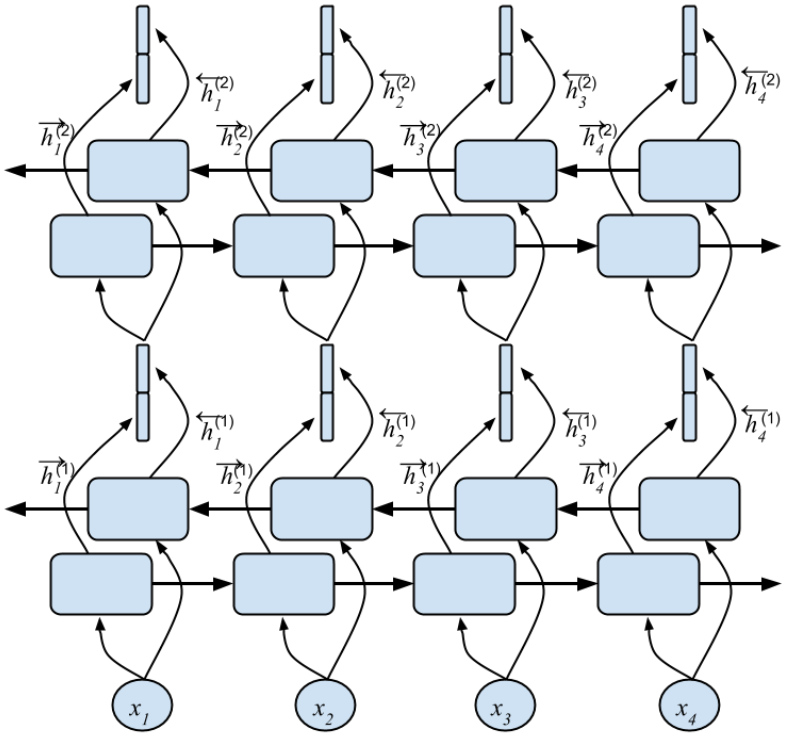

Figure 2.11: Illustration of a stacked bidirectional RNN with two layers.

2.3.3.4 Encoder-Decoder Architecture and Attention

The RNN encoder-decoder architecture was proposed by Cho et al. (2014b) for the task of neural machine translation (NMT). It consists of two RNNs: the first RNN reads the input sequence and encodes it in a single fixed-sized vector, the second RNN receives that vector and generates the output sequence. The two RNNs are trained together to maximise the probability of the target sentence given the source sentence. Figure 2.12 illustrates this process. The encoder RNN encodes the input sequence to a vector:

$$f_{enc\_RNN}(\boldsymbol{x_1}, ..., \boldsymbol{x_T}) = \boldsymbol{c} \tag{2.15}$$

The decoder RNN at every step receives the previously predicted word, the previous state of the decoder and the output of the encoder $\boldsymbol{c}$:

$$s_i = f_{dec\_RNN}(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}) \tag{2.16}$$

Cho et al. (2014b) use RNNs with Gated Recurrent Units (described in Section 2.3.3.2). However, it is possible to use another network as encoder and/or decoder, such as an LSTM (Sutskever et al., 2014) or a CNN (Badrinarayanan et al., 2015).[7]

Cho et al. (2014a) and Bahdanau et al. (2014) argue that encoding a source into one fixed-sized vector might be ineffective on longer sequences. To address this, Bahdanau et al. (2014) propose the **attention** mechanism that allows the decoder to focus on different parts of the encoded sequence at different steps, instead of using one vector as a representation of the source sequence. In this case, the encoder encodes the source into a sequence of states: $\boldsymbol{h}_1, ..., \boldsymbol{h}_T$. The decoder's state is computed similarly to Equation 2.16, with the only difference being that at every

[7] Badrinarayanan et al. (2015) use a CNN to encode and decode images for the task of image segmentation.

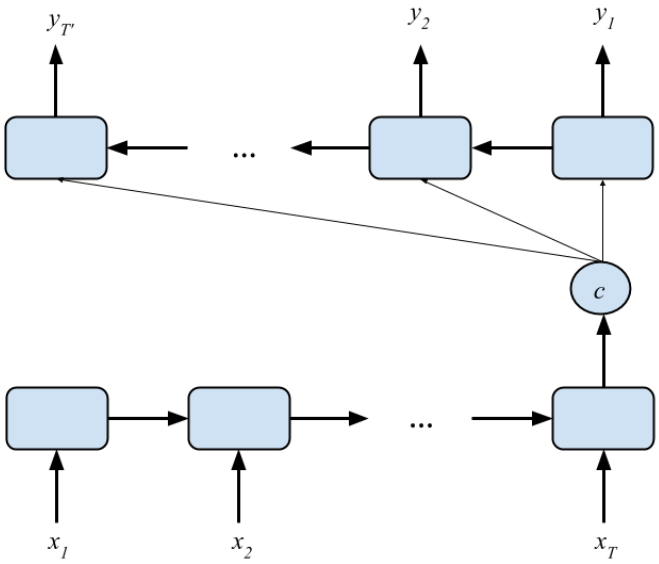

Figure 2.12: Illustration of an RNN encoder-decoder architecture. The encoder RNN reads the input sequence and encodes it in a single fixed-sized vector, the decoder RNN receives that vector and generates the output sequence.

step a distinct context vector is used:

$$s_i = f_{dec\_RNN}(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i) \tag{2.17}$$

The context vector $\boldsymbol{c}_i$ is computed as a weighted sum of the outputs of the encoder:

$$\boldsymbol{c}_i = \sum_{j=1}^{T_x} \alpha_{ij} \boldsymbol{h}_j$$

The weights α_{ij} are computed using the softmax function:

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$$

where e_{ij} represent an alignment model, i.e. how well the inputs around position j

match the output at position i:

$$e_{ij} = \boldsymbol{v}_a^\top \tanh(\boldsymbol{W}_a \boldsymbol{s}_{i-1} + \boldsymbol{U}_a \boldsymbol{h}_j)$$

$\boldsymbol{v}_a, \boldsymbol{W}_a, \boldsymbol{U}_a$ are weights to be tuned. This approach allowed Bahdanau et al. (2014) to improve over the results of Cho et al. (2014b).

Besides the success in NMT, similar attention mechanisms were used in sentiment analysis (Kumar et al., 2016), image caption generation (Xu et al., 2015), object recognition (Mnih et al., 2014) and other tasks.

2.3.4 Other Architectures

There are many other important architectures that we do not describe, because they lie outside the scope of the thesis research. These models include simple and very powerful memory networks (Sukhbaatar et al., 2015), recursive neural networks (Socher et al., 2011), neural Turing machines (Graves et al., 2014) and generative adversarial networks (Goodfellow et al., 2014).

2.4 Training Neural Networks

In Section 2.1 we already described how a logistic classifier is trained. Neural networks are trained in the same way, i.e. by minimising the loss function on the training set. The loss $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ is a function that maps the network outputs $\hat{\boldsymbol{y}}$ and the true labels $\boldsymbol{y}$ to a non-negative scalar representing the error the network makes on the training set. Usually the loss should only be 0 when the output is correct, i.e. $\hat{\boldsymbol{y}} = \boldsymbol{y}$. Common losses for classification are:

- Negative log-likelihood, also known as cross-entropy:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_i y_i \log(\hat{y}_i)$$

which in case of binary classification simplifies to:

$$L(y, \hat{y}) = -\hat{y}\log(y) - (1 - \hat{y})\log(1 - y)$$

- Mean squared error:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_i (y_i - \hat{y}_i)^2$$

- Hinge-loss for binary classification:

$$L(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y})$$

The loss function is usually minimised using gradient descent or its variations. Note, that minimising the loss function means finding the optimal set of network's parameters. We will further denote the loss function as $L(\hat{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{\theta})$ or $L(\boldsymbol{\theta})$, where θ represents all the trainable parameters of the network, such as weights and biases of all the layers of an MLP described in Section 2.3.1. In particular, the gradients of the loss are computed with respect to the trainable parameters, this is done using the **backpropagation** algorithm (Rumelhart et al., 1986): (1) a **forward pass** is performed on the training examples, i.e. the network parameters are considered fixed; (2) the loss is calculated for the obtained predictions; (3) the errors are then propagated backwards starting from the output and desired weight updates are obtained. The recurrent neural networks described in Section 2.3.3 are trained using **backpropagation through time** (BPTT), i.e. backpropagation applied to the unrolled multilayer network.

2.4.1 Parameter Initialisation

Deep learning models are very sensitive to initialisation and it is important to set the initial parameters correctly, so the training can converge and can also avoid getting stuck at a local minimum. According to Goodfellow et al. (2016), neural

network optimisation is not yet fully understood and perhaps the only thing known with certainty is the importance of "breaking symmetry". The latter means that if there are two units with the same activation function connected to the same inputs, they should be initialised differently in order to prevent the network from always keeping them equal to each other. Usually random initialisation solves this issue.

For feedforward neural networks it is advised to set the initial weights to small random values, while the biases should be set to zeros or small positive values (Goodfellow et al., 2016). One way to initialise the weights of a fully connected layer with m inputs and n outputs is to sample the weights randomly from either the uniform distribution: $W \sim U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$ or a truncated normal distribution with 0 mean and standard deviation of $\frac{1}{\sqrt{m}}$: $W \sim \mathcal{N}(0, \frac{1}{m})$. Glorot and Bengio (2010) proposed another heuristic for initialising the weights, what they called normalised initialisation, but what is usually known by the forename of the first author as **xavier initialisation**:

$$W \sim U\Big(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\Big) \tag{2.18}$$

Goodfellow et al. (2016) suggest to initialise biases with zeros.

When initialising the weights of the recurrent neural networks, such as LSTMs, fewer heuristics are available. Many studies (Luong et al., 2015; Sutskever et al., 2014) sample the initial weights from the uniform distribution around zero, i.e. $U(-0.1, 0.1)$.

A strategy to initialise the weights of convolutional networks was derived by He et al. (2015). Their derivations show that the CNNs with ReLU activations should have the initial biases set to zeros, and the weights should be initialised from the normal distribution with zero mean and standard deviation of $\frac{\sqrt{2}}{k\sqrt{c}}$, where k is the filter size and c is the number of channels:

$$\mathbf{W} \sim \mathcal{N}(0, \frac{2}{k^2 c})$$

2.4.2 Stochastic Gradient Descent

In practice, calculating the weight updates on the full training set is very slow, and this is done on small subsets of the training set called **mini-batches** instead. This technique of applying gradient descent to mini-batches is called **stochastic gradient descent (SGD)**. When the subset is equal to only one training example this is sometimes called **online training** mode. The size of the mini-batch is a hyperparameter to be experimentally chosen. There are several extensions of SGD that use adaptive learning rate, e.g. Adagrad (Duchi et al., 2011) and Adam (Kingma and Adam, 2015).

2.4.3 Overfitting and Regularisation

Neural networks are very powerful approximators, and if the architecture and the hyperparameters are chosen correctly, the network is able to fit the training set perfectly. This fact has its own downsides, as the network may lose its ability to generalise, i.e. **overfit** the training set and perform poorly on unseen instances.

One way to prevent overfitting on the training set is to periodically evaluate the model on a held-out set. The training should be stopped when the performance on the held-out set stops improving, i.e. the model starts overfitting the training data. In practice, both training and development losses (or sometimes accuracy, precision or another metric) are measured at every iteration. If the development loss is not decreasing for a number of consecutive iterations (while the training loss keeps decreasing [8]), this is a strong indicator that a model has started overfitting. In this case the training is stopped and the last best (before the development loss stopped decreasing) model is used. This technique is called **early stopping**.

Regularisation is a technique aimed to prevent the network from overfitting the training data. In this work we use the two most common types of regularisation: weight decay and dropout. **Weight decay** is an essential form of regularisation

[8]The training loss should be always decreasing, if the model is learning. If it does not, this may indicate the learning rate is set to a too large a value or there is another problem with the model

that involves adding the sum of the norms of the weights to the loss function, in order to prevent the weights from growing too much. We usually use weight decay with L2 norm, also referred to as **L2 regularisation**.

$$\text{L2\_loss} = ||\boldsymbol{\theta}||_2 = \frac{1}{2}\sum_i \theta_i^2$$

and the loss function in this case takes the following form:

$$loss = L(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{\theta}) + \alpha \text{L2\_loss}$$

where α is the **regularisation rate** which is usually set to a small number [9] that can be tuned on a development set.

Another common form of regularisation is **dropout** (Srivastava et al., 2014). Dropout is a very powerful regularisation technique that consists in *dropping* some units during the training. This is illustrated in Figure 2.13. This prevents the network from relying on a few neurons and forces it to perform more robustly. Note, that the dropout is applied during training only. The dropout assumes that every neuron is activated with a certain probability, i.e. let $\boldsymbol{h} = (h_1, h_2, ..., h_n)$ denote the activations of a layer. Then during the training $\boldsymbol{h}$ is set to $\boldsymbol{h} \star \boldsymbol{b}$, where each dimension of $\boldsymbol{b}$ is sampled from a Bernoulli distribution: $b_i \sim Bernoulli(p)$, where p is a hyperparameter that is often set to 0.2-0.5. During inference, the activations $\boldsymbol{h}$ are scaled by p, i.e. are set to $p\boldsymbol{h}$.

In recurrent neural networks, dropout is usually applied only to the feedforward connections (Pham et al., 2014; Zaremba et al., 2014; Bluche et al., 2015), i.e. to the input and to the output before it is passed to the next step. It was believed that applying dropout to recurrent connections was not desirable. However, a more recent technique by Gal and Ghahramani (2016) suggests that the recurrent connections could also be dropped, but the same mask should be used at each step.

[9] This number should depend on the number of trainable parameters and the size of the training set. However, we did not find any heuristics on setting this in the literature. Discovering an optimal strategy for setting this parameter is perhaps, a good direction for future work.

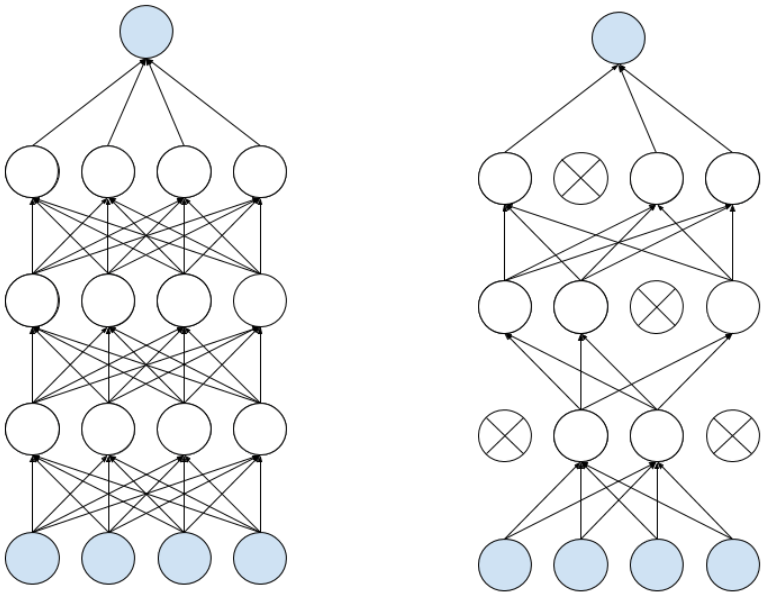

Figure 2.13: A multilayer perceptron before (left) and after (right) dropout. The crossed circles represent dropped units.

2.5 Unsupervised Pretraining

Neural networks are usually powerful enough to learn from raw data, instead of relying on handcrafted features extracted with external tools. It has been shown that NNs can even successfully learn directly from bytes (Gillick et al., 2015). In our case the raw representations are usually words or characters. They are first represented as vectors in a low-dimensional space and are often called **embeddings**. In this thesis we will use the terms word embeddings, word vectors and word representations interchangeably.

A common way to initialise the embeddings used by a neural network is to sample them randomly from a uniform distribution with low standard deviation. Another option is to use pretrained word vectors. Training word embeddings was first suggested by Bengio et al. (2003), however, did not gain popularity at that point. Nowadays, the most widely used models for unsupervised pretraining of word

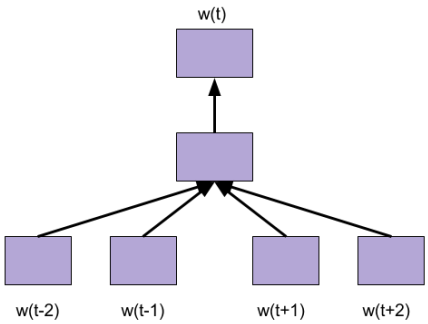

Figure 2.14: CBOW model

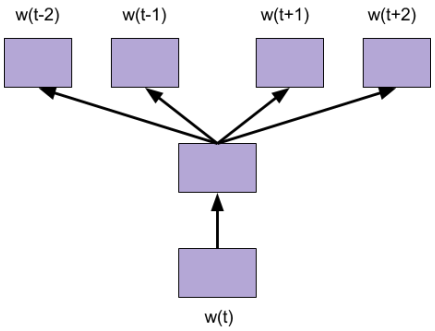

Figure 2.15: Skip-gram model.

embeddings are the skip-gram and the continuous bag of words models presented by Mikolov et al. (2013b), these models are also known as *word2vec*, the name of the software providing the implementation of the two models.[10]

2.5.1 Word Embeddings

The continuous bag of words (CBOW) and the skipgram models presented by Mikolov et al. (2013a) first initialise all word vectors randomly, and the CBOW predicts the word using its context (see Figure 2.14), while the skipgram model does the opposite, given a word, predicts the context around it (see Figure 2.15).

CBOW More formally, let V be the vocabulary size. Let $\boldsymbol{x}_1, ..., \boldsymbol{x}_V$ be one-hot encodings of all words in the vocabulary. The one-hot encodings are mapped to dense vectors using a matrix $\boldsymbol{W}$ of size $V \times d$, which is initialised randomly, and d is the desired dimensionality of the embeddings. The dimensionality d is usually much smaller than the vocabulary size, e.g. it is usually set to a number between 50 and 500. The word embedding for a word $\boldsymbol{x}_i$ could be then denoted as $\boldsymbol{v}_i = \boldsymbol{W}^\top \boldsymbol{x}_i$. Let's assume $w_1, w_2, ...w_T$ to be the sequence of training words. The CBOW model predicts the word given its context, i.e. the k words to the left and the k words to the right from the given word, i.e. the goal of the CBOW model is to maximise the

[10] `https://code.google.com/archive/p/word2vec/`

following probability:

$$p(w_t|w_{t-k}, ..., w_{t-1}, w_{t+1}, ...w_{t+k})$$

Let's consider an example sentence: *The panda eats shoots and leaves*, and let the context window be equal to exactly one word: $k = 1$. The CBOW model predicts the word *panda* using *the* and *eats*; the word *eats* using *panda* and *shoots*, and so on. Figure 2.16 illustrates the CBOW model for the *the panda eats* example, i.e. the prediction of the word *panda* given its context. Each row of the matrix $\boldsymbol{W}$ represents a word in the vocabulary. The vector $\boldsymbol{h}$ is obtained by averaging (or sometimes, concatenating) the input word representations, i.e. for the example sentence:

$$\boldsymbol{h} = \frac{1}{2}\boldsymbol{W}^\top(\boldsymbol{x}_{the} + \boldsymbol{x}_{eats}) = \frac{1}{2}(\boldsymbol{v}_{the} + \boldsymbol{v}_{eats})$$

or in a general case:

$$\boldsymbol{h} = \frac{1}{C}\boldsymbol{W}^\top(\boldsymbol{x}_1 + ... + \boldsymbol{x}_C) = \frac{1}{C}(\boldsymbol{v}_{w_1} + ... + \boldsymbol{v}_{w_C}) \tag{2.19}$$

where $\boldsymbol{x}_1, ..., \boldsymbol{x}_C$ are the context words, and $\boldsymbol{v}_{w_1}, ..., \boldsymbol{v}_{w_C}$ are their embeddings.

$\boldsymbol{W}'$ (see Figures 2.16) is another weights matrix[11] of size $d \times V$, which is used to compute the score for each word in the vocabulary. Let $\boldsymbol{v'}_i$ denote $\boldsymbol{W}'\boldsymbol{x}_i$, then the score u_i for the word w_i is:

$$u_i = \boldsymbol{v'}_i^\top \boldsymbol{h} = (\boldsymbol{W}'\boldsymbol{x}_i)^\top \boldsymbol{h} \tag{2.20}$$

The output word w_o is predicted using softmax:

$$\text{softmax}(u_t) = \frac{e^{(u_t)}}{\sum_{i=1}^{V} e^{(u_i)}} \tag{2.21}$$

[11] we omit biases for simplicity

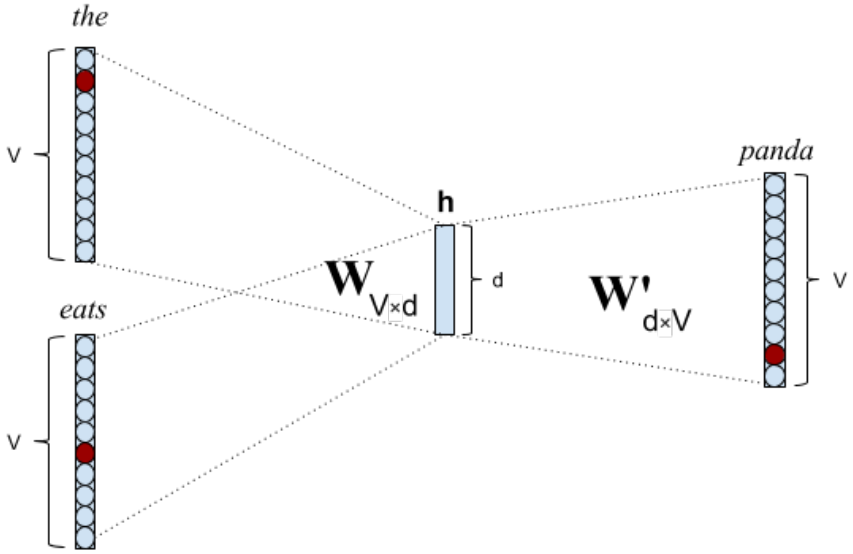

Figure 2.16: Illustration of the CBOW model for the word *panda* in context *the panda eats*. The model predicts the word *panda* using *the* and *eats*.

Skip-gram The skip-gram model is very similar to the CBOW, with the only difference being that instead of predicting the word given its context, it does the opposite, i.e. predicts the context given the word. More formally, the skip-gram model maximises the average log-probability:

$$\frac{1}{T}\sum_{i=1}^{T}\sum_{\substack{-k\leq j\leq k\\ j\neq 0}}\log(w_{t+j}|w_t) \tag{2.22}$$

Let's return to our example sentence: *The panda eats shoots and leaves*. The skip-gram model given the word *panda* and the window $k = 1$ tries to predict the words *the* and *eats*; and given the words *eats* it tries to predict *panda* and *shoots*, and so on. Figure 2.17 illustrates this process. The vector $\boldsymbol{h}$ in Figure 2.17 is the transposed embedding of the input word:

$$\boldsymbol{h} = \boldsymbol{W}^\top \boldsymbol{x}_I = \boldsymbol{v}_{w_I}^\top \tag{2.23}$$

and the score for each output word is predicted using the softmax, as in the Eq. 2.21, i.e. for the example in Figure 2.17, the probability for the *eats* given the word panda

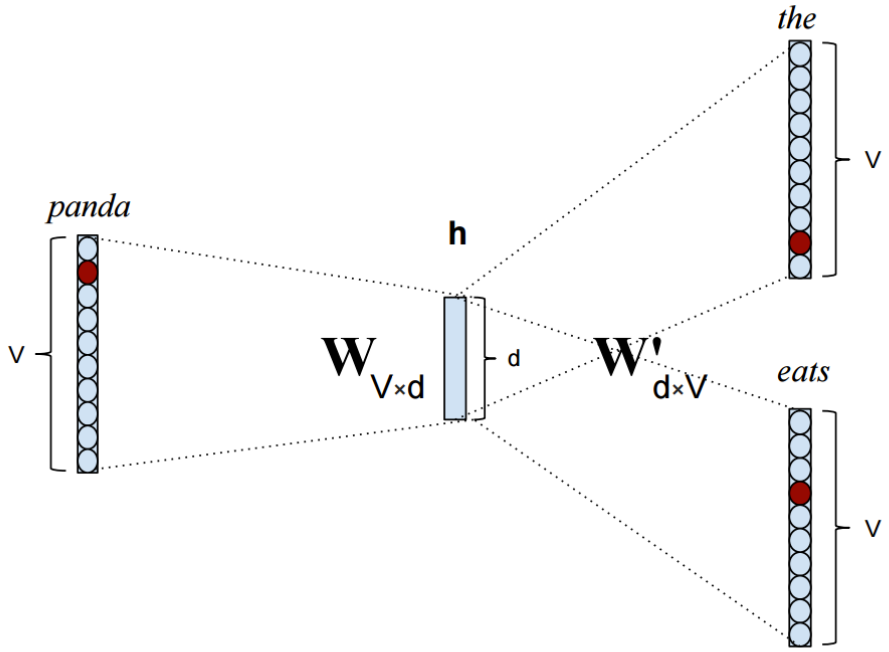

Figure 2.17: Illustration of the skip-gram model predicting the words *the* and *eats* given the word *panda*, i.e. the model given the word *panda* and the window of one word learns to predict the words *the* and *eats*.

is:

$$p(eats|panda) = \frac{e^{\boldsymbol{v'}_{panda}^{\top}\boldsymbol{v}_{eats}}}{\sum_{i=1}^{V} e^{\boldsymbol{v'}_{i}^{\top}\boldsymbol{v}_{eats}}} \tag{2.24}$$

In practice, the softmax over the whole vocabulary is unfeasible. Instead, a technique called **negative sampling** (Gutmann and Hyvärinen, 2012) is used. The idea behind negative sampling is to make sure that the noise can be distinguished from the positive examples using a binary logistic regression. For that, *num_neg* negative (also called contrastive) words w_i^- are sampled. Mikolov et al. (2013c) suggest to sample w_i^- from the unigram distribution raised to the power 3/4, this will ensure that the frequent words are sampled slightly less frequently. The objective function of the negative sampling is defined as follows:

$$L_{NEG} = -\log \sigma(\boldsymbol{v'}_{w_o}^{T}\boldsymbol{h}) - \sum_{i=1}^{num\_neg} \log \sigma(-\boldsymbol{v}_{w_i^-}^{T}\boldsymbol{h}) \tag{2.25}$$

Another way to train these models is in using **hierarchical softmax** (Morin and Bengio, 2005) instead of a full softmax.

The idea behind using a linear classifier for prediction is to reduce the computational complexity and allow for faster training of the embeddings on larger datasets. Even though these embeddings might be not as good as if a non-linear model were used for their training, they are designed to be trained quickly and eventually used as an input to another non-linear model that can fine-tune them for a specific task.

The word embeddings obtained with these models were shown to not only perform well on the task on word semantic similarity, but also capture meaningful semantic and syntactic regularities: for instance, the vector of *king* – *man* + *woman* is very close to the vector of *queen*, and *apple* – *apples* is similar to *cars* – *car* (Mikolov et al., 2013c).

2.6 Hyperparameter Tuning

How the hyperparameters are tuned is perhaps the most obscure area in deep learning. On the one hand, deep neural networks have many hyperparameters and are very sensitive to the settings of some of them. On the other hand, there are few known good strategies to set them. When there are only three or fewer hyperparameters, such as in the case of Support Vector Machines, grid search or random search is the most desirable strategy. However, when we deal with dozens of hyperparameters, which is the case with complex neural architectures, this becomes infeasible. A recent paper by Zoph and Le (2016) of the Google Brain team reports to have used 800 GPUs to find the best settings. There are also a few studies that suggest using Bayesian optimisation for hyperparameter tuning (Snoek et al., 2012; Eggensperger et al., 2013).

The most feasible approach is still manual search.[12] However, in this case, a good understanding of the role of each hyperparameter is needed. In this section we

[12]There are jokes suggesting an alternative name to intelligent manual search: GSD a.k.a. *Graduate Student Descent* (Kevin Duh at DL4MT winter school)

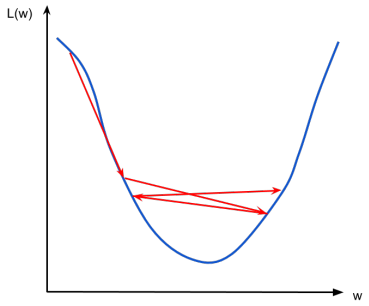

Figure 2.18: Learning with too high learning rate.

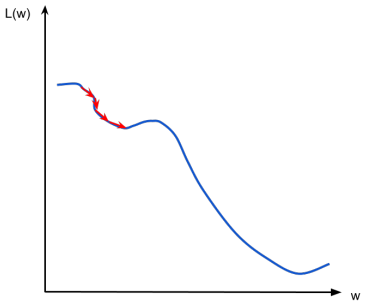

Figure 2.19: Learning with too low learning rate.

will review the main hyperparameters of the architectures we use.

Learning rate: When we train a model with a gradient-based method, such as stochastic gradient descent, we update the trainable parameters **w** as follows:

$$w_i = w_i - \lambda \frac{\partial L(\boldsymbol{w})}{\partial w_i} \tag{2.26}$$

where $L(w)$ is the loss function, and λ is the **learning rate**. It determines how far down the direction opposite to the gradient we want to move the weights. This is perhaps the most important hyperparameter to set correctly. When it is set to too large a value, we risk simply missing the optimal point (see Figure 2.18). If the learning rate is too small, the training becomes slower and we risk getting stuck at a local minimum (see Figure 2.19). That is why it is important to try a range of values when setting the learning rate.

Hyperparameters that increase the representation capacity: most parameters directly influence the model's representation capacity. For example, increasing the **number of hidden layers** makes the model able to learn more complicated functions. The same happens when we increase the number of hidden units or the dimensionality of the word embeddings. The size of the convolution kernel falls into the same group. There are also binary hyperparameters, such as the use of a

bidirectional RNN instead of a unidirectional RNN, and the use of LSTM instead of GRU.

Increasing the values of these hyperparameters increases the number of trainable parameters of the model, i.e. it can better represent the training set. Ideally, the values of these hyperparameters should be just as large as needed. In practice, it is very hard to find settings that match the problem's complexity exactly. Thus, our general approach to tuning these parameters is to have them slightly higher than needed, and then add strong regularisation. Having these parameters set at higher values than actually needed makes the model prone to overfitting on the training set. In other words, we let the model be able to overfit the training set and then regularise it with weight decay and dropout.

Regularisation Hyperparameters: these hyperparameters increase the model's representation capacity when decreased, i.e. having the dropout probability and the weight decay set to very low values allows the model fit the training set better, but also decreases its ability to generalise. Usually for a complex model we set the dropout probability to 0.3 or 0.5 and the weight decay rate, e.g. L2 regularisation rate, to a small value around 10^{-7}–10^{-5}. We increase the weight decay rate if we use a very deep network, and decrease it for smaller networks.

Other hyperparameters: other parameters include the choice of the activation functions and the choice of the optimiser (SGD versus its variations). In our experiments we usually use ReLU activation for the feedforward networks and hyperbolic tangent for the recurrent network, and plain SGD optimiser.

Training parameters: Apart from the model hyperparameters, there are a few other parameters that affect the training process, such as the size of the minibatch and the number of training steps. We usually determine the number of training steps with early stopping. We need to set the following parameters: after how many iterations do we evaluate the model on the development set, and how long do we

wait for an improvement. In our experiments, we set the number of iterations after which we evaluate the model on the development set roughly equal to the number of iterations needed to iterate over the whole training set once. We call this number an epoch. We usually wait for ten consecutive epochs for an improvement on the development set, then we stop the training.

2.7 Summary

Deep learning is an area of machine learning that specialises in models with several layers of nonlinear units, allowing higher representation capacity and multiple levels of abstraction. In the area of Natural Language Processing, deep learning gained its popularity mainly because it obviates the need of feature engineering and allows us to learn from raw representations. Despite being very powerful and achieving state-of-the-art results in many tasks, it is sometimes criticised for the lack of interpretability and some attempts are being made to decipher the models (Li et al., 2016).

In this chapter we have summarised the basics of deep learning in application to some natural language processing tasks. We only reviewed the neural architectures we will need to describe the experiments presented in this thesis. A comprehensive overview of deep learning techniques can be found in the Deep Learning book by Goodfellow et al. (2016). New architectures and techniques appear almost every day, for most recent advances in deep learning we advise the reader to check `arXiv.org` and recent conference proceedings, e.g. Conference on Neural Information Processing Systems (NIPS)[13] and International Conference on Learning Representations (ICLR)[14].

[13] `https://www.nips.cc/`
[14] `http://www.iclr.cc/`

Chapter 3

Detecting Semantically Equivalent Questions in CQAs

This chapter describes our early work on the task of question-question semantic similarity. The work described in this chapter was carried out during an internship at IBM Research Brazil between September and December 2014 in collaboration with Cicero dos Santos. Since 2014, deep learning techniques have advanced enormously. At the end of this chapter, we reconsider the experimental setup and the findings of this work.

As we compare two questions with a view to provide the answer to one of them, in this context *similar* questions means that these questions have the same answer. This chapter focuses on the task of question classification, i.e. given a pair of questions, predicting if they can be answered with the same answer.

The focus of this work is in learning from naturally annotated data of CQA websites. In particular, in most of our experiments we use data from Stack Exchange[1] communities. This community advises its users to search the forum for an answer before posting a new question, as it might already have been asked. However, finding this question is not always a straightforward task, as different users can formulate the same question in completely different ways. The Stack Exchange community

[1] `http://stackexchange.com`

Title: I can't download anything and I can't watch videos	**Title:** How can I install Windows software or games?
Body: Two days ago I tried to download skype and it says an error occurred it says *end of central directory signature not found Either this file is not a zipfile, or it constitutes one disk of a multi-part archive. In the latter case the central directory and zipfile comment will be found on the last disk(s) of this archive. zipinfo: cannot find zipfile directory in one of* `~/Downloads/SkypeSetup-aoc-jd.exe` or `~/Downloads/SkypeSetup-aoc-jd.exe.zip`, and cannot find `~/Downloads/SkypeSetup-aoc-jd.exe.ZIP` pe... this happens whenever I try to download anything like games and also i can't watch videoss it's looking for plug ins but it doesn't find them i hate this [sic!]	**Body:** Can *.exe* and *.msi* files (Windows software) be installed in Ubuntu? [sic!]
Link: `http://askubuntu.com/questions/364350`	`http://askubuntu.com/questions/988`
Possible Answer (Shortened version): .exe files are not binary-compatible with Ubuntu. There are, however, compatibility layers for Linux, such as Wine, that are capable of running .exe.	

Table 3.1: An example of semantically equivalent questions from Ask Ubuntu community.

and some other user forums have a duplication policy. Exact duplicates, such as copy-and-paste questions, and nearly exact duplicates are usually quickly detected, closed and removed from the forum. Nevertheless, some duplicate questions are kept. The main reason for that is that *there are many ways to ask the same question, and a user might not be able to find the answer if they are asking it a different way.*[2]

We define two questions as **semantically equivalent** if they can be adequately answered by the exact same answer. Table 3.1 presents an example of a pair of such questions from Ask Ubuntu forum. Detecting semantically equivalent questions is a very difficult task for two reasons: (1) the same question can be phrased in many different ways; and (2) two questions can be asking different things but looking for the same solution. Therefore, traditional similarity measures based on word overlap

[2] `http://stackoverflow.com/help/duplicates`

such as shingling and Jaccard coefficient (Broder, 1997) and its variations (Wu et al., 2011) are not able to capture many cases of semantic equivalence.

This chapter is structured as follows: Section 3.1 addresses the task of question classification. In particular, in Section 3.2 we introduce our methodology, including the neural architecture used in most experiments. Then, we describe our experimental setup in Section 3.3 including the datasets and the baselines. We experimentally compare the neural architecture to the baselines in Section 3.4. We compare the use of domain-specific versus out-of-domain word embeddings in Section 3.5. We measure the impact of the training set size on the performance in Section 3.6. We evaluate our approach on another domain in Section 3.7. We provide an error analysis in Section 3.8. As this chapter presents experiments done at early stages of the research, in Section 3.9 we reexamine the experimental setup of this chapter and suggest how it can be improved. In Section 3.10 we provide an overview of approaches to the task of question retrieval in community question answering websites. We draw our conclusions in Section 3.11.

3.1 Question Classification Task

Following the duplication policy of the Stack Exchange online community,[3] we define semantically equivalent questions as follows:

Definition 3.1.1. Two questions are semantically equivalent if they can be adequately answered by the exact same answer.

Since our definition of semantically equivalent questions corresponds to the rules of the Stack Exchange duplication policy, we assume that all questions of this community that were marked as duplicates are semantically equivalent.[4] An example

[3] `http://blog.stackoverflow.com/2010/11/dr-strangedupe-or-how-i-learned-to-stop-worrying-and-love-duplication/`; `http://meta.stackexchange.com/questions/32311/do-not-delete-good-duplicates`

[4] This assumption does not always hold true in reality, as was later reported by Hoogeveen et al. (2016)

of such questions is given in Table 3.1. These questions vary significantly in vocabulary, style, length and content quality. However, both questions require the exact same answer.

The exact task that we approach in this section is a binary classification task: given two problem definitions, predict if they are semantically equivalent. By *problem definition* we mean the concatenation of the title and the body of a question. Throughout this chapter we use the term *question* as a synonym of problem definition.

3.2 Methodology

In this section we use a convolutional neural network architecture to detect semantically equivalent questions. The words are first transformed into word embeddings, using a large collection of unlabelled data, and then we apply a convolutional neural network (CNN, described in Section 2.3.2) to build distributed vector representations for pairs of questions. Finally, the questions are scored using a similarity metric. Pairs of questions with a similarity score above a threshold, defined based on a held-out set, are considered duplicates. The CNN is trained using positive (semantically equivalent) and negative (not semantically equivalent) pairs of questions. During training, CNN is *induced* to produce similar vector representations for questions that are semantically equivalent.

We perform experiments using data from two different Stack Exchange forums. We compare CNN performance with a Support Vector Machines classifier (Cortes and Vapnik, 1995) and a duplicate detection approach based on shingling (Broder, 1997).

We also investigate the performance of the network in different settings. In particular, we evaluate:

- the impact of word embeddings pretrained on in-domain data versus the ones pretrained on the English Wikipedia;

- word vectors of different dimensionalities;
- the influence of the training set size on the performance of the CNN versus the baseline system;
- the impact of in-domain word embeddings when using out-of-domain training data.

3.2.1 Neural Network Architecture

The encoder we use to obtain the representations for questions must deal with two main challenges: first, different questions can have different sizes; and second, important information can appear at any place in the question, i.e. we can not cut off the end of a question. The convolutional approach (Waibel et al., 1989) is one of the ways to tackle these challenges. In recent work, convolutional approaches have been used to solve similar problems when creating representations for text segments of different sizes (dos Santos and Gatti, 2014) and character-level representations of words of different sizes (dos Santos and Zadrozny, 2014).

As detailed in Figure 3.1, a convolutional neural network (see Section 2.3.2 for more details on this architecture) is used to encode each of the two questions, i.e. the concatenations of their title and body. The input to the network is the tokenised question text. First, the words are transformed into word embeddings of size d, where d is a hyperparameter to be chosen. The word embeddings form the only input channel of the CNN. Then a convolutional layer produces local features around each word in the question: the filter (word window) of size w is used, where w is another parameter to be experimentally tuned. The network then combines these local features using a sum operation (similar to max pooling described in Section 2.3.2) to create a fixed-sized vector representation for the question: $\boldsymbol{r}_{q_1}$ for the first question and $\boldsymbol{r}_{q_2}$ for the second question.

Finally, the CNN computes a similarity score between $\boldsymbol{r}_{q_1}$ and $\boldsymbol{r}_{q_2}$. In our exper-

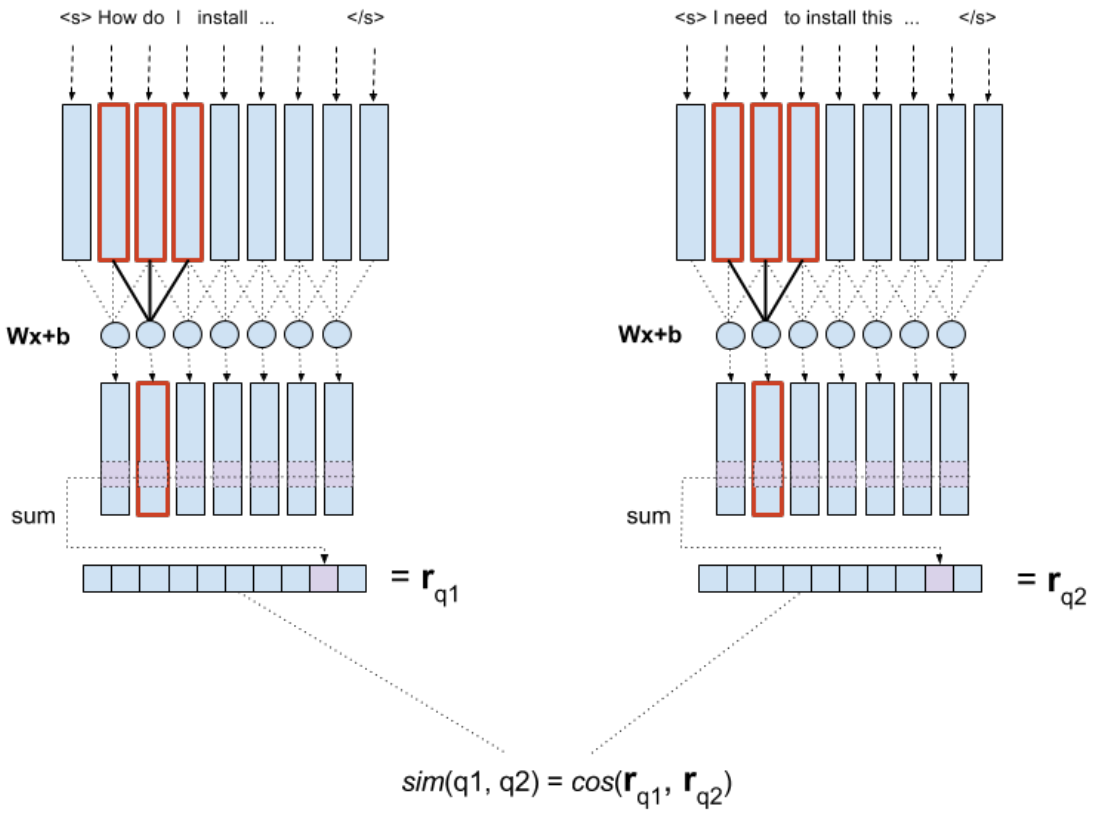

Figure 3.1: Convolutional neural network for semantically equivalent questions detection.

iments we use the cosine similarity

$$s(q_1, q_2) = \frac{\boldsymbol{r}_{q_1} \cdot \boldsymbol{r}_{q_2}}{|\boldsymbol{r}_{q_1}||\boldsymbol{r}_{q_2}|}$$

Pairs of questions with similarity above a threshold, defined based on a heldout set, are considered duplicates.

The network is trained by minimising the mean-squared error over the training set. Given a question pair (q_1, q_2), the network with parameter set $\boldsymbol{\theta}$ computes a similarity score $s_\theta(q_1, q_2)$. Let $y_{(q_1,q_2)}$ be the correct *label* of the pair, where its possible values are 1 (equivalent questions) or 0 (not equivalent questions). We use stochastic gradient descent (SGD) to minimize the mean-squared error with respect to θ:

$$L(\theta) = \sum_{(x,y)\in D} \frac{1}{2} \left(y - s_\theta(x)\right)^2 \tag{3.1}$$

where $x = (q_1, q_2)$ corresponds to a question pair in the training set D and y represents its respective label $y_{(q_1,q_2)}$.

3.3 Experimental Setup

3.3.1 Datasets

In our experiments we use data from the Ask Ubuntu Community Questions and Answers (Q&A) site.[5] Ask Ubuntu is a community for Ubuntu users and developers, and it is part of the Stack Exchange[6] Q&A communities. The users of these communities can ask and answer questions, and vote up and down both questions and answers. Users with a high reputation become moderators and can label a new question as a duplicate to an existing question.[7] Usually it takes five votes from different moderators to close a question as a duplicate.

We use the Ask Ubuntu data dump provided in May 2014. We extract all question pairs linked as duplicates. The data dump we use contains 15277 such pairs. For our experiments, we randomly select a training set of 24K pairs, a test set of 6K and a development set of 1K, making sure there are no overlaps between the sets. Half of each set contains pairs of semantically equivalent questions (positive pairs) and half are pairs of questions that are not semantically equivalent. The latter pairs are randomly selected from the corpus.

For the experiments on a different domain that we report in Section 3.7 we use the Meta Stack Exchange[8] data dump provided in September 2014. Meta Stack Exchange (Meta) is used to discuss the Stack Exchange community itself. People ask questions about the rules, features and possible bugs. The data dump we use contains 67746 questions, where 19456 are marked as duplicates. For the experiments on this data set, we select random balanced disjoint sets of 20K pairs for training,

[5] `http://askubuntu.com/`

[6] `http://stackexchange.com`

[7] More information about Stack Exchange communities could be found here: `http://stackexchange.com/tour`

[8] `http://meta.stackexchange.com`

1K for development and 4K for testing. We prepare the data in exactly the same manner as the Ask Ubuntu data.

3.3.2 Baselines

We explore three main baselines: a method based on the Jaccard coefficient which was reported to provide high accuracy for the task of duplicate detection (Wu et al., 2011), a Support Vector Machines (SVM) classifier (Cortes and Vapnik, 1995) and the combination of the two.

Shingling baseline: for this baseline, the documents are first represented as sets of shingles, i.e. unique n-grams, of lengths from one to four, and then the Jaccard coefficient for a pair of documents is calculated as follows:

$$J(S(d_1), S(d_2)) = \frac{S(d_1) \cap S(d_2)}{S(d_1) \cup S(d_2)},$$

where $S(d_i)$ is the set of shingles generated from the ith document. High values of the Jaccard coefficient denote high similarity between the documents. If the value exceeds a threshold T, the documents are considered semantically equivalent. In this case, the training data is used to select the optimal threshold T.

SVM baseline: for the SVM baseline, we represent the documents with n-grams of length up to four. For each pair of questions and each n-gram we generate three features:

1. if the n-gram is present in the first question;
2. if the n-gram is present in the second question;
3. the overall normalised count of the n-gram in the two questions.

We use the Radial Basis Function (RBF) kernel and perform grid search to optimize the values of C and γ parameters.[9] We use a frequency threshold to reduce the

[9]We used the script provided with the libsvm library that explores C in $[2^{-5}; 2^{15}]$ and γ in $[2^{-15}; 2^{3}]$

number of features.[10] The implementation provided by LibSVM (Chang and Lin, 2011) is used.

Combined baseline: in order to combine the two baselines, for a pair of questions we calculate the values of the Jaccard coefficient with shingles size up to four, and then add these values as additional features used by the SVM classifier.

3.3.3 Word Embeddings

The word embeddings used in our experiments are initialised by means of unsupervised pretraining. We perform pretraining using the skip-gram architecture (Mikolov et al., 2013c) available in the word2vec[11] tool. Two different corpora are used to train word embeddings for most of the experiments: the English Wikipedia and the Ask Ubuntu community data. The experiments presented in Section 3.7 also use word embeddings trained on the Meta Stack Exchange community data. In the experiments with the English Wikipedia word embeddings, we use the embeddings previously produced by dos Santos and Gatti (2014). They have used the December 2013 snapshot of the English Wikipedia corpus to obtain word embeddings with the skip-gram model of the word2vec tool.

In the experiments with the Ask Ubuntu and the Meta Stack Exchange, we use the Stack Exchange data dump provided in May 2014 to train the word embeddings. Three main steps are used to process all questions and answers from these Stack Exchange dumps: (1) tokenisation; (2) image removal, URL replacement and prefixing/removal of the code if necessary (see Section 3.4 for more information); (3) lower-casing of all tokens.

The resulting corpora contains about 121 million and 19 million tokens for Ask Ubuntu and Meta Stack Exchange, respectively.

[10]Several values (2, 5, 35 and 100) were tried with cross-validation, the threshold with value 5 was selected

[11] `http://code.google.com/p/word2vec/`

3.3.4 Hyperparameters

The cosine similarity threshold was set to 0.5. The window size is set to 3, the number of convolutional units is set to 300, and the word embeddings size to 200 (we experiment with different word embedding dimensionalities in Section 3.5). The learning rate was experimentally set to 0.005. Out-of-vocabulary words, i.e. those not present in the training set or whose frequency in the training set did not exceed the threshold of 5, are mapped to the `<UNK>` token.

3.4 Comparison with Baselines

In this section we experimentally compare the performance of the CNN versus the performance of the baseline systems.

Code Handling: the Ask Ubuntu community gives users an opportunity to format parts of their posts as code by using *code* tags (an example is in italic in Table 3.1). It includes not only programming code, but commands, paths to directories, names of packages, error messages and links. Around 30% of all posts in the data dump contain *code* tags. Since the rules for code formatting are not well defined, it was not clear if a learning algorithm would benefit from including it or not. Therefore, for each algorithm we tested three different approaches to handling code: keeping it as text; removing it; and prefixing it with a special tag. The latter is done in order to distinguish between the same term used within text or within code or a command (e.g., a *for* as a preposition and a *for* in a for loop). When creating the word embeddings, the same approach to the code as for the training data was followed.

The development set is used to tune the hyperparameters of the algorithms. In order to speed up computations, we perform our initial experiments using a 4K balanced subset of the training set.

We test the shingling-based approach with different shingle sizes. As Table 3.2

indicates, the accuracy decreases with the increase of the shingle size. The fact that much better accuracy is achieved when comparing questions based on simple word overlap (shingle size 1), suggests that semantically equivalent questions are not duplicates but rather have topical similarity. The SVM baseline performs well only when combined with the shingling approach by using the values of the Jaccard coefficient for shingle size up to four as additional features. A possible reason for this is that n-gram representations do not capture enough information about semantic equivalence. The CNN with word embeddings outperforms the baselines by a significant margin.

The results presented in Table 3.2 indicate that the algorithms do not benefit from including the code. This is probably because the code tags are not always used appropriately and some code examples include long error messages, which make the user generated data even more noisy. Therefore, in the following experiments the code is removed. Perhaps, learning separate representations for code could have been advantageous for this task.

The development accuracy and the test accuracy using the full 24K training set is presented in Table 3.3. The SVM with four additional shingling features is found best among the baselines (see Table 3.2) and is used as a baseline in this experiment. Again, the CNN with word embeddings outperforms the best baseline by a significant margin.

3.5 Impact of Domain-Specific Word Embeddings

We perform two experiments to evaluate the impact of the word embeddings on the CNN accuracy. In the first experiment, we gradually increase the dimensionality of word embeddings from 50 to 400. The results are presented in Figure 3.2. The vertical axis corresponds to development accuracy and the horizontal axis represents the training time in epochs. As has been shown by Mikolov et al. (2013c), word embeddings of higher dimensionality trained on a large enough data set capture semantic

Approach	Features	Code	Dev Acc.	Hyperparameters
SVM-RBF	binary + freq.	kept	66.20	C=8.0, γ ≈3.05e-05
SVM-RBF	binary + freq.	removed	66.53	C=2.0, γ ≈1.2e-04
SVM-RBF	binary + freq.	prefixed	66.53	C=8.0, γ ≈3.05e-05
Shingling size 1	-	kept	72.35	-
Shingling size 1	-	removed	72.65	-
Shingling size 1	-	prefixed	70.94	-
Shingling size 2	-	kept	69.24	-
Shingling size 2	-	removed	66.83	-
Shingling size 2	-	prefixed	67.74	-
Shingling size 3	-	kept	65.23	-
Shingling size 3	-	removed	62.93	-
Shingling size 3	-	prefixed	64.43	-
SVM-RBF	bin+freq+shing	kept	74.0	C=32.0, γ ≈3.05e-05
SVM-RBF	bin+freq+shing	removed	**77.4**	C=32.0, γ ≈3.05e-05
SVM-RBF	bin+freq+shing	prefixed	73.6	C=32.0, γ ≈3.05e-05
CNN	AU word vectors	kept	91.3	w=3, d=200,
CNN	AU word vectors	removed	**92.4**	conv. units=300
CNN	AU word vectors	prefixed	91.4	learning rate=0.005

Table 3.2: Development Accuracy and best parameters for the baselines and the Convolutional Neural Network.

System	Dev Acc.	Test Acc.
SVM + shingles	85.5	82.4
CNN + Askubuntu	**93.4**	**92.9**

Table 3.3: CNN and SVM accuracy on the development and the test set using the full training set.

information better than those of a smaller dimensionality. The experimental results presented in Figure 3.2 corroborate these findings: we can see improvements in the neural network performance when increasing the word embeddings dimensionality from 50 to 100 and from 100 to 200. However, increasing the dimensionality from 200 to 400 does not improve the performance significantly enough to justify the increase in the training time.

In the second experiment, we evaluate the impact of in-domain word embeddings on the network's performance. We obtain word embeddings trained on two different corpora: the Ask Ubuntu community data and the English Wikipedia (see Section 3.3.3). Both representations are 200-dimensional. The results presented in

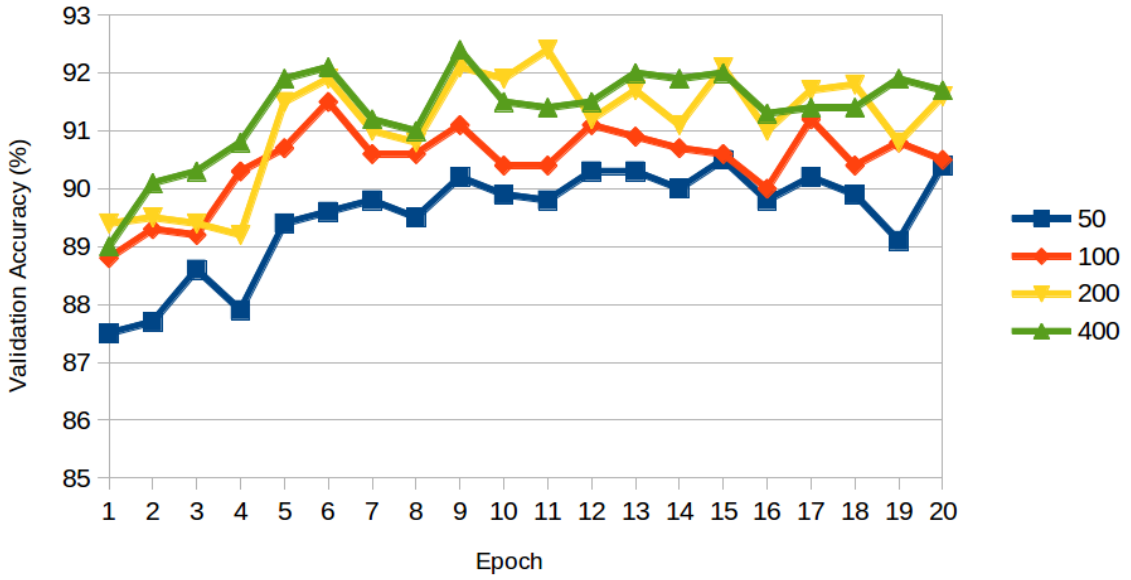

Figure 3.2: CNN accuracy depending on the size of word embeddings

Word Embeddings	# tokens	Dev Acc.
Wikipedia	≈1.6B	85.5
Ask Ubuntu	≈121M	**92.4**

Table 3.4: Development Accuracy of the CNN with word embeddings pretrained on different corpora.

Table 3.4 show that training on in-domain data is more beneficial for the network, even though the corpus used to create word embeddings is much smaller.

3.6 Impact of Training Set Size

In order to measure the impact of the training set size, we perform experiments using subsets of the training data, starting from 100 question pairs and gradually increasing the size to the full 24K training set.[12] Figure 3.3 compares the learning curves for the SVM baseline (with parameters and features described in Section 3.4) and for the CNN with word embeddings trained on Ask Ubuntu and English Wikipedia. The vertical axis corresponds to the development accuracy, and the horizontal axis represents the training set size. As Figure 3.3 indicates, increasing the size of the

[12] We use sets of 100, 1000, 4000, 12000 and 24000 question pairs.

training set provides improvements. Nonetheless, the difference in accuracy when training with the full 24K training set and 4K subset is about 9% for SVM and only about 1% for the CNN. This difference is small for both word embeddings pretrained on Ask Ubuntu and Wikipedia but, the in-domain word embeddings provide better performance independent of the training set size.

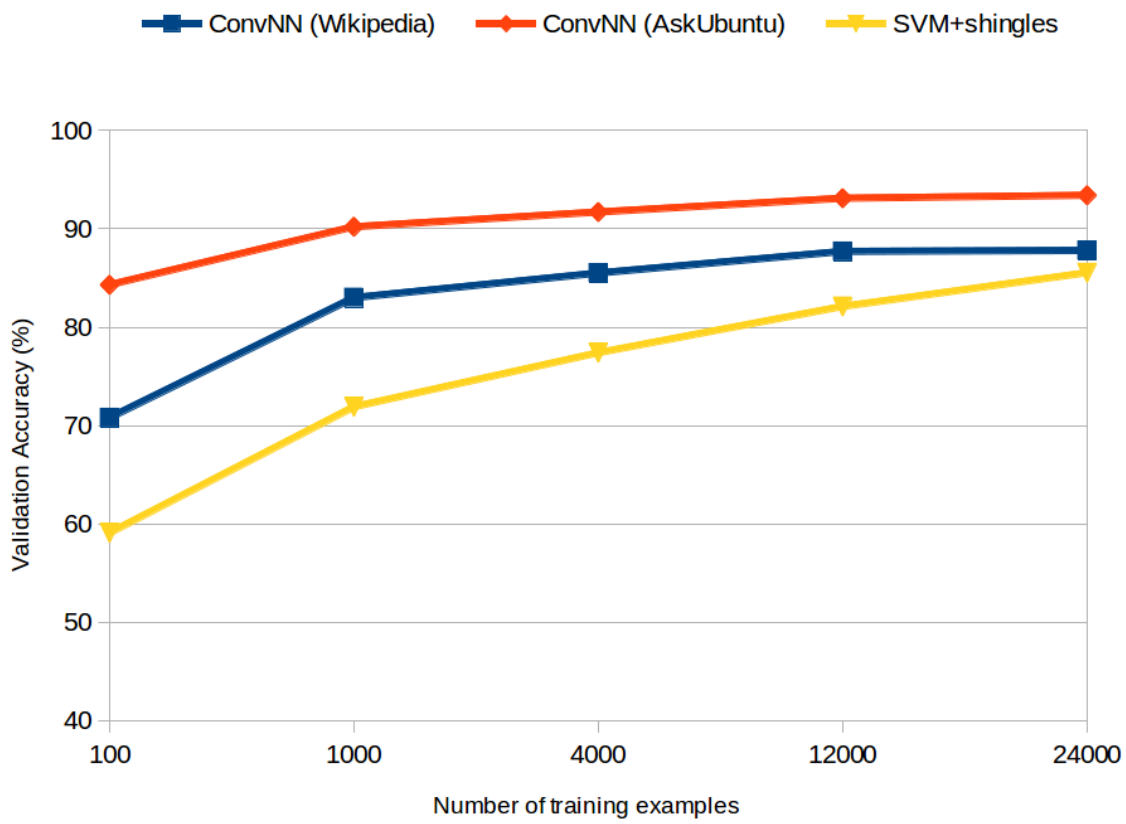

Figure 3.3: Development accuracy for the baseline and the CNN depending on the size of training set.

3.7 Experiments on a Different Domain

Muthmann and Petrova (2014) report that the Meta Stack Exchange Community[13] is one of the hardest for finding semantically equivalent questions. We perform the same experiments described in previous sections using the Meta data set. In Table 3.5, we can see that the CNN accuracy on Meta test data (92.68%) is similar to the one for the Ask Ubuntu community on test data (92.4%) (see Table 3.3).

[13] http://meta.stackexchange.com/

Training Data	Size	Word Embeddings	Dev Acc.	Test Acc.
META	4K	META	91.10	89.97
META	4K	Wikipedia	86.90	86.27
META	20K	META	**92.80**	**92.68**
META	20K	Wikipedia	90.60	90.52
Ask Ubuntu	24K	META	83.90	83.35
Ask Ubuntu	24K	Ask Ubuntu	76.8	80.00

Table 3.5: Convolutional Neural Network Accuracy tested on Meta Stack Exchange community data.

Also, in Table 3.5, we show results of a domain adaptation experiment in which we do not use training data from the Meta forum. In this case, the CNN is trained using Ask Ubuntu data only. The numbers show that even in this case using in-domain word embeddings helps to achieve relatively high accuracy: 83.35% on the test set. The performance is about 3% lower when both the word embeddings and the training data come from another domain.

3.8 Error Analysis

The convolutional neural network with in-domain word embeddings outperforms the baselines based on lexical features in the task of identifying semantically equivalent questions. Even being semantically equivalent, most questions are too different in terms of vocabulary, and in this case the use of CNN is more beneficial than the vocabulary-based methods. The error analysis shows that the CNN is better at distinguishing questions with similar vocabulary but different meanings. For example, the question pair, (q_1) *How can I install Ubuntu without removing Windows?* and (q_2) *How do I upgrade from x86 to x64 without losing settings?* is erroneously predicted as a positive pair by the SVM classifier, while the CNN classifies it correctly as a negative pair.

There are some cases where both CNN and SVM fail to identify semantic equivalence. Some of these cases include questions where essential information is presented as an image, e.g., a screenshot, which was removed during preprocessing.[14] Future

[14]For instance, `http://askubuntu.com/questions/450843`

research could involve the use of a multi-modal system to address this issue.

3.9 The Chapter Reexamined

This chapter described preliminary research carried out in 2014. In this section, we summarise some drawbacks of this work and suggest how this could be improved.

Solving a real world task: In this chapter we have addressed the task of binary classification of question pairs. This task is of a rather synthetic nature, because in the real world scenario one would need to solve the harder task of question retrieval: given a question and a collection of existing questions with their answers, the goal is to retrieve existing questions semantically equivalent to the new one. In the next section we will briefly review existing approaches to this task.

Creating a more realistic dataset: The dataset we used for classification is balanced, i.e. 50% of the pairs were semantically equivalent, and 50% were not. Even though this is a common way to create datasets for binary classification tasks, e.g. a recent dataset of duplicate questions from Quora,[15] this simplifies the task, as semantically equivalent questions are usually much more rare than ones that are not semantically equivalent. That is why creating a dataset with a realistic distribution of semantically equivalent questions would help to better estimate the quality of the evaluated methods, as in many cases skewed datasets represent a challenge for classification methods.

Improving the neural architecture: We used a simple convolutional neural network to encode the questions, and then compared them using cosine similarity. The network's good performance was likely due to the quality of the word embeddings it used. Since 2014 the field has moved forward, suggesting many possible ways to improve the architecture. First of all, several different filter

[15] `https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`

sizes can be used, as well as better regularisation techniques, such as weight decay and dropout (see Chapter 2 for details on these techniques). Besides, as we deal with long questions, a recurrent neural network encoder would probably be more suitable in this case (see our findings in Chapter 6 for the task of answer ranking).

3.10 Related Work on Question Retrieval

This chapter presented an approach to detection of semantically equivalent questions, suggesting a model for a rather synthetic task of binary classification of question pairs. The end goal of semantically equivalent question detection is question retrieval: given a new question retrieve existing previously asked questions that are semantically equivalent to it. In this section we summarise previous attempts to solve the task of question retrieval. We roughly divide the approaches into three groups: (1) early approaches based on rules and templates; (2) approaches applying statistical techniques, including classical term-weighting models such as Okapi BM25 (Robertson et al., 1994) and its variations; and (3) representation learning approaches. The rest of this section describes the related work on each of the four approaches more in detail.

3.10.1 Rules and Templates for Question Retrieval

Some early systems used templates to match new questions to existing ones. Katz (1997) described START, the first online question answering system that used rules to convert a question to a ternary expression in the form `<subject relation object>`. For instance, these rules aimed to provide the system with information on the equivalence of *A surprised B with C* and *A's C surprised B*. The rules were represented as `if-then` statements: `If <<subject surprise object1> with object2> Then <object2 surprise object1>`.

Another example of template-based question matching is the work of Sneiders

(2002). They proposed a system for answering one-sentence FAQ-style factoid questions by introducing question templates – parametrised questions with entity slots, e.g. *What is the best restaurant in* `<CITY>`*?* A new question was then matched to existing templates.

A way to provide better templates for matching questions is to use knowledge bases to find synonyms. For example, Burke et al. (1997) extracted synonyms and hypernyms from WordNet (Miller, 1995) to better match questions to questions and answers from FAQ archives.

3.10.2 Statistical Techniques for Question Retrieval

Most existing approaches to question retrieval adapt existing information retrieval methods to the task. The adaptations aim to overcome two issues: (1) similar questions being very different on the lexical and syntactic levels (2) questions being short, which turns out to be an issue, as most retrieval models, e.g. one of the most popular retrieval models Okapi BM25 (Robertson et al., 1994), were designed for the retrieval of long documents.

In order to overcome the lexical gap issue, many studies including Zhou et al. (2011) and Cai et al. (2011) turn to the idea of Berger and Lafferty (1999), who suggested using statistical machine translation methods to bridge the lexical gap between the query and the document. In particular, they use IBM Model 1 (Brown et al., 1993) to estimate word-to-word translation probabilities.

Most studies suggest using translation models together with the language models (Jeon et al., 2005; Xue et al., 2008; Duan et al., 2008; Zhou et al., 2011). Given a question q and a candidate question q_c, the score of the question q_c with respect to the query q is estimated as the following probability:

$$p_{q|q_c} = \prod_{w \in q} p(w|q_c)$$

where word probabilities $p(w|q_c)$ are provided by both a translation model and

a language model:

$$p(w|q_c) = \lambda p_{LM}(w|q_c) + (1 - \lambda)p_{TM}(w|q_c)$$

p_{LM} and p_{TM} are the probabilities defined by the language model and the translation model respectively, and λ is a parameter that controls the effect of each of the models.

Several studies further improve on this approach, e.g. Zhou et al. (2011) incorporate phrase-level information into this model instead of relying only on the word translation probabilities; Cai et al. (2011) and Zhang et al. (2014) suggest using Latent Dirichlet Allocation (Blei et al., 2003) to discover the latent topic information in the questions. The former uses only the questions in the retrieval, while the latter uses the answers too. Both Cai et al. (2011) and Zhang et al. (2014) combine the topical information with the translation-based retrieval language (TRLM) model (i.e. the combination of the translation model and the language model described above):

$$p(w|q_c) = \lambda p_{TRLM}(w|q_c) + (1 - \lambda)p_{LDA}(w|q_c)$$

To address the second issue, i.e. short document retrieval, Wang et al. (2009) and Zhang et al. (2012) suggest using syntactic information. Zhang et al. (2012) parse the questions into undirected dependency graphs, and then estimate the distances between terms as the length of the path between them in the graph. They define the *dependency relevance* between two terms as follows:

$$dep(t_1, t_2) = \frac{1}{b^{path(t_1,t_2)}}$$

where $b > 1$ is a hyperparameter that is tuned to maximise mean average precision. They define the term weights to be a linear combination of the similarities defined by pointwise mutual information and the dependency relevance. The weights obtained in this way are then multiplied by the weights of BM25, translation-based or another

retrieval model.

3.10.3 Representation Learning for Question Retrieval

With the recent success of representation learning and deep learning in natural language processing (see Chapter 2 for an overview), research on question retrieval started to focus on representation learning approaches to the task.

One of the first attempts to learn representations for question retrieval was presented by dos Santos et al. (2015). They propose an extension of the work presented in this chapter: the score of the question pair is determined by the combination of the scores provided by the CNN and the score provided by the weighted bag-of-words model. The weighted bag-of-words model represents the questions as sparse vectors of frequencies of each word in the vocabulary, and the final representation of the model is obtained by multiplying this sparse vector by the vector of weights. The latter is optimised during training.

Similarly to the approach we presented in this chapter and the approach of dos Santos et al. (2015), Das et al. (2016) use two convolutional neural networks with shared parameters to represent the questions and then compare the representations using cosine similarity. The architecture is almost identical to ours, except for the additional feedforward layer they incorporate in their network and using different hyperparameters. The network is trained in the same way as the network presented in this chapter, i.e. by maximising the similarity between semantically equivalent questions on the training set, although using a different loss function (they use max-margin loss in case of non-equivalent questions and the error otherwise). In order to use this model in the retrieval setting, they combine the score produced by this model with a score produced by Okapi BM25 model:

$$score = \alpha * score_{CNN} + (1 - \alpha) * score_{BM25}$$

Another model for question retrieval that also combines the representation learn-

ing with the BM25 model was presented by Lei et al. (2016). They propose a novel neural architecture to represent questions. The architecture is essentially a convolutional neural network with an additional gate that is used to downweight certain bigrams. Similarly to Das et al. (2016), they combine the score produced by this model with the score of the BM25 model.

In Chapter 4 we return to the representation learning approaches to community question answering tasks including question retrieval, and give a more thorough overview of the techniques used in this task.

3.11 Summary

In this section we introduced a method for identifying semantically equivalent questions based on a convolutional neural network. We experimentally showed that the proposed CNN achieves very high accuracy especially when the word embeddings are pretrained on in-domain data. The performance of an SVM-based approach on this task was shown to depend highly on the size of the training data. In contrast, the CNN with in-domain word embeddings provides very high performance even with limited training data. Furthermore, experiments on a different domain have demonstrated that the neural network achieves high accuracy independently of the domain.

The results show that:

- using word embeddings pretrained on domain-specific data allows the network to achieve very high performance;
- increasing the dimensionality of word embeddings results in higher accuracy;
- in-domain word embeddings provide better performance even with a smaller training set;
- a convolutional neural network with in-domain word embeddings achieves relatively high accuracy even when using out-of-domain training data.

Chapter 4

Learning to Rank Answers

The goal of the **answer ranking** task is to rank answers according to how well they answer a given question. As this is usually the second step after either question matching or candidate answer selection, this task is sometimes referred to as **answer reranking**.

We focus on the community question answering websites, that serve as the source of labelled data. We assume that the answer selected by the community as the best one (see Figure 4.1) is the one that has to be ranked before all other answers. We will mainly focus on deep learning approaches to answer ranking. We also investigate the possibility of combining the neural approach with handcrafted features based on discourse markers. In this chapter we introduce our general methodology and describe the experimental setup, and devote the next three chapters to the experiments.

This chapter is structured as follows: Section 4.1 presents the related work. In Section 4.2, we describe the ranking problem, the task of answer ranking and our general approach to the task. In Section 4.3 we introduce the datasets we use in answer ranking experiments. Section 4.4 describes the experimental setup including the evaluation metrics and the baselines.

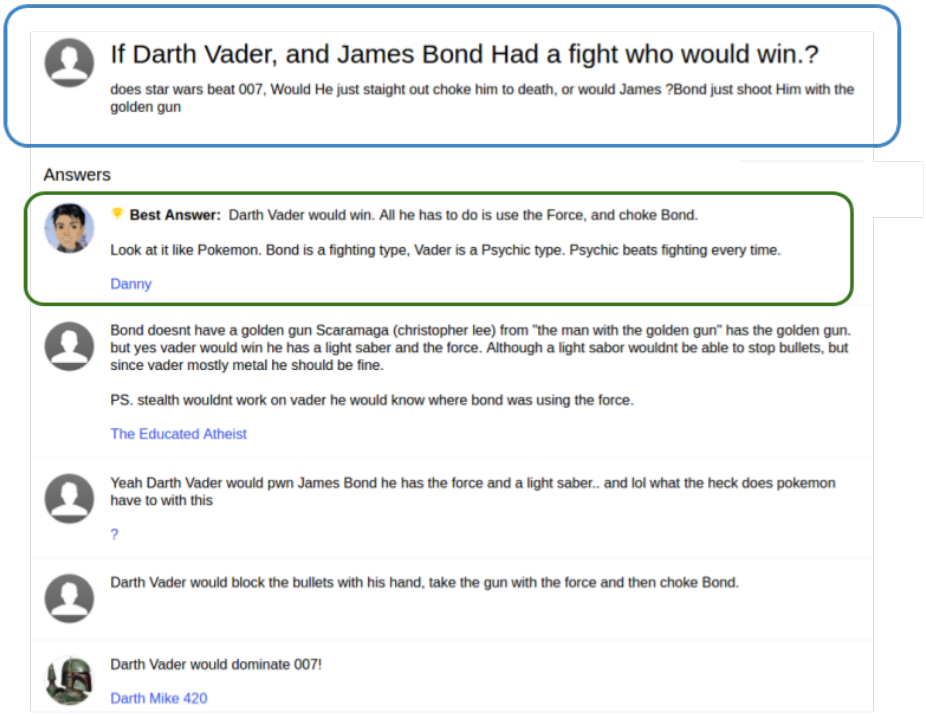

Figure 4.1: Example of a question posted on Yahoo! Answers CQA. The best answer considered as the positive example is in the green box.

4.1 Related Work

4.1.1 Non-Factoid Answer Ranking

One of the main components of a non-factoid question answering system is the answer (re)ranking module. Given a question, it aims to rearrange the answers in order to boost the correct answers to the top positions, or the community-selected best answer in case of the CQA-based AR.

One of the main problems of non-factoid question answering and non-factoid answer ranking in particular is the so called *lexical chasm* (Berger et al., 2000), i.e., questions and answers usually have a very low or no vocabulary overlap at all. Most studies are focused on bridging this lexical chasm. One way to bridge the lexical chasm is to apply Statistical Machine Translation (SMT) techniques. For instance, Riezler et al. (2007) trained an SMT model on a large corpus of question-answer

pairs extracted from FAQ pages, and query expansion was performed by adding best translations of the query, which is used to retrieve answer passages. Another approach to non-factoid answer ranking is to apply a machine learning ranking algorithm to the representations of answers. The answers can be represented with a variety of features including the ones provided by translation models (Surdeanu et al., 2011) as well as many other features such as discourse markers (Jansen et al., 2014) and features based on IR ranking models (Surdeanu et al., 2008). More recent neural approaches learn distributed representations for questions and answers instead of (dos Santos et al., 2016) or together with (Cheng et al., 2016) handcrafted features.

In the next sections we will focus on machine learning methods for answer ranking in CQAs, as this is the main focus of our work. We will roughly divide the methods into feature-based approaches, i.e. where question-answer pairs are represented as vectors of handcrafted features, and the success of the ranking/selection is due to the quality of these features; and neural approaches, where the representations for question-answer pairs are learned and the main contribution is the architecture of the machine learning predictor.

4.1.2 Ranking Scenarios in CQAs

When ranking answers in CQAs, there are three possible scenarios depending on the source of the answers to be ranked:

Ranking answers in a thread: Most studies consider only answers posted in the same thread as the question (Jansen et al., 2014; Fried et al., 2015; Tymoshenko et al., 2016a; Barrón-Cedeño et al., 2015). This means that there are only a few (usually no more than twenty) answers to rank, and this allows *expensive* ranking methods (that require training and/or feature engineering) to be used directly on all possible answers without any pre-selection. Feature-based studies use SVMrank (Joachims, 2006) or Support Vector Regression (Drucker

et al., 1997) as the predictor. Neural approaches often calculate cosine similarity between the learned question and answer representations (dos Santos et al., 2016; Tan et al., 2015).

Ranking answers to related questions: Semeval 2016 Task 3 introduced a subtask on ranking answers that do not belong to the same thread as the given question. Instead, a set of related threads is retrieved using a search engine, and the answers in these threads are to be ranked according to their relevance to the original question (Nakov et al., 2016b). The organisers made available the ranks of each of the related questions provided by the Google search engine. Most participants including the winning team (Mihaylova et al., 2016) approached the task by ranking the answers within each thread (i.e. the previous scenario when only the answers within the same thread are ranked) and then multiplying the scores by the reciprocal rank of the respective related question. A different method was used by Nakov et al. (2016a). They consider pairwise interactions between the original question, the related question and the answer to the related question. They extract the features for the three possible pairs and then use a supervised predictor.

Ranking a larger collection of answers from a CQA: Surdeanu et al. (2011) and Hieber and Riezler (2011) rank all community-selected best answers (to many different questions). This means ranking thousands of answers for every question, which is prohibitive for the direct use of *expensive* machine learning techniques. This task is usually addressed in two steps: first, a list of candidate answers is retrieved using a *cheap* unsupervised model such as tf-idf (Salton and McGill, 1986) or Okapi BM25 (Robertson et al., 1994). Then a handful of top answers is re-ranked, for instance, using a feature-based predictor, such as SVMRank (Joachims, 2006), SVR (Drucker et al., 1997) or Ranking Perceptron (Shen and Joshi, 2005).

4.1.3 Features for Non-Factoid Answer Ranking

In this section we will outline different types of features used for the task of answer ranking in CQA, as well as several related tasks, including non-factoid question answering and answer quality prediction in CQAs. In order to structure the previous work on the task, we will roughly classify the features into character-level, lexical, syntactic, semantic, discourse and non-textual features.[1]

4.1.3.1 Character-Level Features

A few studies use character-level information in order to extract features from questions and answers. For instance, Hieber and Riezler (2011) measure the informativeness of the text as the entropy of the character distribution (as well as the word distribution). Toba et al. (2014) uses several features indicating whether and how often non-ASCII or special punctuation symbols are present in the question and/or the answer, assuming that a good answer is not likely to have a high proportion of special symbols. Barrón-Cedeño et al. (2015) use nine heuristic binary features indicating whether the answer contains certain symbols, e.g. *@* or *?*. They also check if the answer includes three or more consecutive repeated characters, e.g. *aaaaa*.

4.1.3.2 Lexical Features

Most studies on non-factoid answer ranking use lexical features, i.e. features relying on word-level representations of the text. These features are computed on either bag-of-word representations, or bag-of-n-gram representations.

Lexical Similarity Features: These features measure similarity between a question and an answer on a lexical level. Several studies calculate normalised word and n-gram overlaps between the question and the answer (Toba et al., 2014; Yi et al., 2015), expecting to have a sufficient lexical overlap between a

[1]Note that this categorisation may not be perfectly precise as some features may belong to several categories (e.g. Surdeanu et al. (2011) apply BM25 similarity not only to lexical representations but also to syntactic dependencies and semantic roles).

question and an answer, despite the known *lexical chasm* issue. Other similarity measures include the length-normalised BM25 formula (Robertson et al., 1994) that was used by Surdeanu et al. (2008, 2011) as follows:

$$BM25(A) = \sum_{i=0}^{|Q|} \frac{(k_1+1)tf_i^A(k_3+1)tf_i^Q}{(K+tf_i^A)(k_3+tf_i^Q)} \log(idf_i) \tag{4.1}$$

where Q is the question, A is the answer, tf_i^A and tf_i^Q are the frequencies of the question term i in A and Q respectively, idf_i is the inverse document frequency of the term i in the whole corpus; and K is the length normaliser:

$$K = k_1((1-b) + b|A|/ans\_len)$$

where ans_len is the average answer length in the collection. b, k_1 and k_3 are the parameters of the BM25 model, that are usually set as follows: $k_1, k_3 \in [1.2; 2.0]$, $b = 0.75$.

Another common way to measure the similarity between the question and the answer is in measuring the Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951) between unigram language models for the question and the answer (Agichtein et al., 2008; Liu et al., 2008; Cong et al., 2008; Wang et al., 2010). Let M_u and M_v be two language models, then the KL-divergence is estimated as follows:

$$D_{KL}(M_u||M_v) = \sum_w p(w|M_u) \log \frac{p(w|M_u)}{p(w|M_v)} \tag{4.2}$$

Other ways to measure lexical similarity include estimating n-gram co-ocurrence statistics by adapting measures for machine translation evaluation, e.g. Soricut and Brill (2004) use BLEU (Papineni et al., 2002) to estimate the n-gram overlap between the question and the answer; calculating Jaccard similarity (Jenders et al., 2016) between n-gram representations (as we do in Chapter 3 for

the task of duplicate question detection); cosine similarity between *tf-idf* and other lexical representations of the question and the answer (Cong et al., 2008; Jansen et al., 2014).

Density and Frequency Features: These features measure the frequency and density of question words in the answer: e.g. the number of non-stop question words that appear in the answer (Surdeanu et al., 2011); and *answer span*, i.e. the largest distance between two question words in the answer (Hieber and Riezler, 2011; Surdeanu et al., 2011). In several studies density and frequency features are applied not only on the lexical level but also on several other levels of text representation (e.g. Toba et al. (2014) calculates the density of punctuation signs and non-ASCII symbols in the answer, and Surdeanu et al. (2011) and Hieber and Riezler (2011) estimate informativeness of the answer based on the frequencies of various part of speech tags).

Features Extracted with Gazetteers and Regular Expressions: These features are usually binary features indicating whether a question or an answer contains a word from a particular gazetteer (a list of words of a dictionary) or match a certain regular expression. For instance, Yi et al. (2015) use a binary feature indicating if any of the question words, i.e. *why*, *what*, *when*, *how*, *which* etc., is present in the answer, as this may indicate the answer is only asking a follow-up question, and thus, is not informative.

Several studies create gazetteers for their tasks. For instance, Verberne et al. (2011) in their study on answering *why* questions created a list of 47 of words "that introduce the explanation", e.g. *because*, *as a result of*, *which explains why*. Higashinaka and Isozaki (2008) uses the Japan Electronic Dictionary Research Institute dictionary[2] to create a list of word pairs, with a causality relation between them, i.e. *crime* and *arrest*.

Regular expressions are mostly used to detect if the answer contains a URL or

[2] `http://www2.nict.go.jp/r/r312/EDR/index.htm`

an e-mail (Barrón-Cedeño et al., 2015) or to count them in the answer (Toba et al., 2014).

4.1.3.3 Syntax-Level Features

POS tags: These features measure frequencies of different POS tags (Hou et al., 2015) and overlap between a question and an answer in terms of POS tags (Yi et al., 2015; Verberne et al., 2011). In addition, Hieber and Riezler (2011) measure the **formality** of a text as the proportion of nouns, adjectives, prepositions and articles, against the pronouns, verbs, adverbs and interjections; and **informativeness** as the number of non-stop nouns, verbs and adjectives that are present in the answer but not in the question.

Syntactic Trees: Several studies represent the question and the answer as a bag of syntactic dependencies (Surdeanu et al., 2011; Fried et al., 2015). Surdeanu et al. (2011) compute the BM25 similarity not only on the lexical but also on the level of syntactic dependencies. Toba et al. (2014) manually prepare a list of syntactic patterns to determine the **focus word** of *what* and *which* questions. The focus word is the main focus of the question, e.g. in *What is your favourite movie?* the focus word is *movie*. They also create a set of patterns to identify the **focus adjective** for *how* questions, e.g. in *How old are you?* the focus adjective is *old*. Verberne et al. (2010) hypothesise that certain syntactic parts of a question are more important when deciding if a particular answer is good for that question. They divide the question into the following syntactic structures: heads, phrase modifiers; the subject, main verb, nominal predicate, and direct object of the main clause; and all noun phrases. Then they calculate the word overlap between each of these syntactic constituents and the answer. For some of these features they also indicate if the overlapping word is used with the same syntactic function in both the question and the answer.

Barrón-Cedeño et al. (2015) represent the question and the answer as shallow constituency trees in order to estimate the similarity between questions and answers with partial tree kernels (Moschitti, 2006). Tymoshenko et al. (2016b) use shallow syntactic trees with an additional **REL** tag pre-pended to the nodes that contain the words shared by the question and the answer and also encode the question subject as a separate subtree with a **SUBJECT-S** tag under the **ROOT** node.

Readability and Grammaticality: Several studies measure the syntactic and semantic complexity of the answer. For instance, Agichtein et al. (2008) suggest that representing the text as a bag of POS n-grams helps to estimate the level of grammatical quality of the text. The intuition behind this measure is that certain patterns, such as *how/why/when to* usually indicate a lower-quality question while *how/why/when VERB PRONOUN VERB* indicates a higher-quality question (e.g. *how to remove ... ?* versus *how do I remove ... ?*). Hieber and Riezler (2011) measure grammaticality by counting how many word n-grams appear more than three times in the text. Several studies also implement readability measures (Toba et al., 2014; Hieber and Riezler, 2011), most of which are based on the proportion of words with more than three syllables and the sentence length. For instance, the Gunning fog index of readability (Gunning, 1952) is measured as follows:

$$S_{GF} = 0.4(\text{avg sentence length} + \text{proportion of complex words})$$

where complex words are usually words with three or more syllables.

4.1.3.4 Semantic Features

Semantic Analysis: Several studies use various techniques for semantic analysis in order to represent the question and the answer. Inspired by the intuition that a question and its answer should share the same topics, Tran et al. (2015) used

Latent Dirichlet Allocation (Blei et al., 2003) (LDA) to represent the question and the answer as vectors, and then estimated the cosine similarity between these vectors. Nicosia et al. (2015) use Latent Semantic Analysis (Deerwester et al., 1990) and Yang et al. (2016) use Explicit Semantic Analysis (Gabrilovich and Markovitch, 2009) in a similar manner. Some more recent methods use predictive methods for obtaining word representations, e.g. Jansen et al. (2014) and Nakov et al. (2016a) represent the question and the answer as averaged representations obtained by training a skip-gram model (Mikolov et al., 2013b) for learning word embeddings (see Chapter 2 for more details on the skip-gram model).

Semantic Role Labelling: Surdeanu et al. (2011) represent the question and the answers as bags of predicate-argument relations extracted with the semantic parser of Surdeanu and Ciaramita (2007) that uses PropBank notation (Palmer et al., 2005). The output of the parser is converted to semantic dependencies by extracting a dependency between each predicate and every one of its arguments. These dependencies are labelled with the corresponding argument. For instance, for a sentence *A helicopter gets its power from rotors or blades.* the following semantic dependencies are extracted: $gets \xrightarrow{\text{agent}} helicopter$, $gets \xrightarrow{\text{patient}} its\ power$, $gets \xrightarrow{\text{instrument}} from\ rotors\ or\ blades$.

Higashinaka and Isozaki (2008) used several binary features indicating whether a certain causality relation pattern is present in the answer. They used a corpus of Japanese sentences annotated with semantic relations in order to extract these patterns.

WordNet Synsets and Semantic Similarity: Verberne et al. (2010, 2011) link the words to their WordNet synsets and measure the overlap in terms of synsets. They also estimate Lesk semantic relatedness measure (Lesk, 1986) with WordNet (Pedersen et al., 2004), i.e. calculate word overlap between words glosses and estimate the similarity between the question and the answer

as the average pairwise word similarity.

4.1.3.5 Discourse Features

Verberne et al. (2007) were first to show the utility of discourse information in answering non-factoid questions. They prepared a set of *why* questions using the Rhetorical Structure Theory (RST) Discourse Treebank (Carlson et al., 2001). The RST Treebank contains 385 Wall Street Journal articles from the Penn Treebank (Marcus et al., 1993) that were manually annotated. The discourse structure was represented as a tree with elementary discourse units as leaves, and with internal nodes representing text spans. The nodes were also annotated with rhetorical relations.

Carlson et al. (2001) define 78 fine-grained relations that could be grouped into 16 coarse-grained classes. The classes include *Attribution*, *Background*, *Cause*, *Comparison*, *Contrast*, *Explanation*; and for instance, the *Explanation* class includes the relations of *evidence*, *argumentative explanation* and *reason*. They also used the presence of discourse markers indicating explanation as binary features. The discourse markers included *because*, *as a result of*, *which explains why*. Later, Jansen et al. (2014) extended their discourse markers model. They used a more complete list of markers, and each answer was searched for these markers. Each marker divided the answer into two discourse arguments. They labelled every argument with either `QSEG` indicating a substantial overlap between the argument and the question or `OTHER` otherwise. Thus, their features partially lexicalised with the discourse markers (e.g. `QSEG by OTHER`) were more expressive than the binary features of Verberne et al. (2007). Moreover, they also assigned values to each feature using a lexical semantics model provided by *tf-idf* representations or averaged *skip-gram* (Mikolov et al., 2013b) representations. We use these model in Section 7.2 and describe them in more detail there. Jansen et al. (2014) also used discourse parse trees in their model, following (Verberne et al., 2007), although they use the automatic discourse parser of Feng and Hirst (2012).

Study	Questions	Task	Features	Predictor	Data	Size/Split
Higashinaka and Isozaki (2008)	Why	AR	Gaz, SRL, Synt	SMVRank	WHYQA	1000Q, c-v
Verberne et al. (2010)	Why	AR	Synt, Para, Gaz	LR	Webclopedia, Wikipedia	186Q, c-v
Verberne et al. (2011)	Why	AR	Synt, Gaz, Sim	SVM-Rank, SVR, LR, GA, NB	Webclopedia, Wikipedia	186Q, c-v
Surdeanu et al. (2011)	How	AR	POS, Synt, Rel, NE, Gaz, Sem, SMT, Cooc, Dens	RP, SVM-Rank over BM25 results	Yahoo! Answers	142.6K QA pairs, 60/20/20 split
Hieber and Riezler (2011)	How	AR	Sim, SMT, Read, POS, Gram, Entropy	ranking SVM with SGD	Yahoo! Answers	142.6K QA pairs, 60/20/20
Toba et al. (2014)	factoid, opinion, procedure, reason, yes-no	AQ	Synt, Sim, Stats, Stop, POS, SMT, Link, Sent, Dens	Hierarchy of SVM and LR classifiers	Yahoo! Answers	5854Q, c-v
Jansen et al. (2014)	How, Why	AR	Sim, Disc, Synt, Emb	SVM-Rank	Yahoo! Answers; Biology Text Book Why	10K, 50/25/25; 185 how + 193 why, c-v
Fried et al. (2015)	How	AR	Sim, Synt, SMT, Emb	SVM-Rank	Yahoo! Answers	10K, 50/25/25
Barrón-Cedeño et al. (2015)	CQA	AQ	Meta, Sim, Synt	SVM, SVM^{hmm}, CRF, LOR	Qatar Living Forum	2.6K/300/ 329 QA
Jenders et al. (2016)	CQA	AR	Meta, Stat	RF, NB	openHPI forum	835Q, c-v

Table 4.1: Summary of some of the related work. **Task**: AR - answer ranking/selection, AQ - answer quality prediction. **Features:** Sim - lexical similarity features; Stats - statistical features; Stop - stop words; Dens - density features; Cooc - co-occurrences statistics; Para - paraphrases; POS - features based on POS tags; Synt - features based on syntactic trees; NE - named entities; Disc - discourse features; Gaz - Gazetteers and dictionaries including WordNet; Sem - Semantic Roles; Meta - meta-information; SMT - statistical machine translation techniques; Sent - sentiment polarity; Link - presence of links or emails; Readability - scores of readability models; Gram - grammaticality, punctuation, OOV words; Entropy - entropy of the character or word distributions; Emb - word embeddings; TM - topic modelling features. **Predictor**: LR - logistic regression, GA - genetic algorithm (Goldberg and Holland, 1988), NB - Naive Bayes, RP - ranking perceptron, LOR - logistic ordinal regression, RF - random forest classifier. **Split:** Q - question, c-v - cross-validation.

4.1.3.6 Translation Probabilities:

Several studies including Riezler et al. (2007); Surdeanu et al. (2008, 2011); Hieber and Riezler (2011) and Toba et al. (2014) suggest to *bridge the lexical chasm* between questions and answers by applying machine translation techniques, i.e. computing word alignment between questions and answers. Hieber and Riezler (2011) and Surdeanu et al. (2011) estimate the probability of the question Q being a translation of the answer A using IBM Model 1 (Brown et al., 1993):

$$P(Q|A) = \prod_{q \in Q} P(q|A) \tag{4.3}$$

$$P(q|A) = (1 - \lambda)P_{ml}(q|A) + \lambda P_{ml}(q|C) \tag{4.4}$$

$$P_{ml}(q|A) = \sum_{a \in A}(T(q|a)P_{ml}(a|A)) \tag{4.5}$$

where C is the entire collection of the answers, λ is a smoothing parameter, P_{ml} are estimated using maximum likelihood, and $T(q|a)$ refers to the word alignment usually computed by applying the Expectation-Maximisation (EM) (Dempster et al., 1977) algorithm implemented in the GIZA++ tool (Och and Ney, 2003). Hieber and Riezler (2011) estimate the translation probability (Eq. 4.3) on a lexical level, while Surdeanu et al. (2011) estimate this probability not only on a lexical level but also over labelled syntactic dependencies as well as over labelled semantic dependencies. These probabilities are then used as features in their ranking models. Riezler et al. (2007) use the translation model to get most probable translations for question words and use them for query expansion. Fried et al. (2015) estimate the alignments $T(q|a)$ for each pair of words in the vocabulary, and obtain the following vector representations for each word in the vocabulary:

$$\boldsymbol{w} = (T(w|w_1), T(w|w_2), ..., T(w|w_V))$$

where V is the vocabulary size. Then they compare a pair of words by calculating Jensen-Shannon divergence, a finite and symmetric variation of KL divergence. In order to estimate the similarity between the question and the answer, they estimate average, minimum, and maximum pairwise similarities between their words. These similarities are then used as features.

4.1.3.7 Statistical and Non-Textual Features

A number of studies use various sentence-level, word-level and character-level statistical features including question and answer lengths in terms of words (Toba et al., 2014) or sentences (Yi et al., 2015); the number of capitalised words (Hou et al., 2015), whether the answer contains an image (Nakov et al., 2016a).

Many studies rely on the forum's meta-information. For instance, Jenders et al. (2016) in their study of answering questions on a forum of a platform for online courses, use the number of courses visited by the author of the answer as a feature. Nakov et al. (2016a) and Tymoshenko et al. (2016a) check whether the answer is written by the same author as the question, as it is unlikely for a user to answer their own question. Agichtein et al. (2008) use the answer author's user profile to estimate how likely this user is to post a good answer; Barrón-Cedeño et al. (2015) use the question category, e.g. *Socialising*, *Travel*.

4.1.4 Feature-Based Predictors

Having in mind the problem of non-factoid question answering with CQAs, i.e. answering a question with the best possible user-provided answer, we formulate the problem of answer ranking as a ranking task, following Surdeanu et al. (2008, 2011); Jansen et al. (2014); Fried et al. (2015). However, it is also possible to address this problem as a classification task, i.e. predicting whether an answer is good or bad (Barrón-Cedeño et al., 2015) or whether the answer is of a high, medium or low quality (Agichtein et al., 2008).

Several feature-based studies report that their approach is agnostic to the choice

of the ranking method (Surdeanu et al., 2008, 2011; Higashinaka and Isozaki, 2008) as their main contribution is in the features. Most approaches to NF AR that are not using CQAs usually use small manually built datasets where efficiency is not a big concern. They use a supervised predictor such as SVMRank (Joachims, 2006) or a logistic regression classifier directly on the feature representations of the candidate answers (Higashinaka and Isozaki, 2008; Verberne et al., 2007, 2010, 2011). Table 4.1 summarises some of the related work on this task.

4.1.5 Beyond Feature-Based Approaches

In the previous section we reviewed various feature-based approaches to non-factoid answer ranking, i.e. the ranking was achieved due to powerful handcrafted feature representations. In this section we will review the approaches that achieve the ranking by learning representations for questions and answers instead of relying on handcrafted features. These representations are used in combination with a supervised predictor. The representations and the ranking (or labels) are often trained simultaneously. These approaches usually use neural networks to learn representations and/or the ranking, and so we will refer to them as *neural* or the deep learning approaches, as opposed to the feature-based ones.

As deep learning is a relatively recent trend in the NLP community, there have been only a few neural approaches focusing on the particular task of non-factoid answer ranking or selection. Moreover, neural approaches usually do not rely on handcrafted features, it makes them applicable to other tasks. In this section, we will review neural systems for answer selection, including some of the neural approaches to factoid answer selection.

Most deep learning systems to answer selection, for both factoid and non-factoid questions, include all or some of the following components, that we illustrate in Figure 4.2:

Unsupervised Pretraining of the Word Embeddings: The question and the

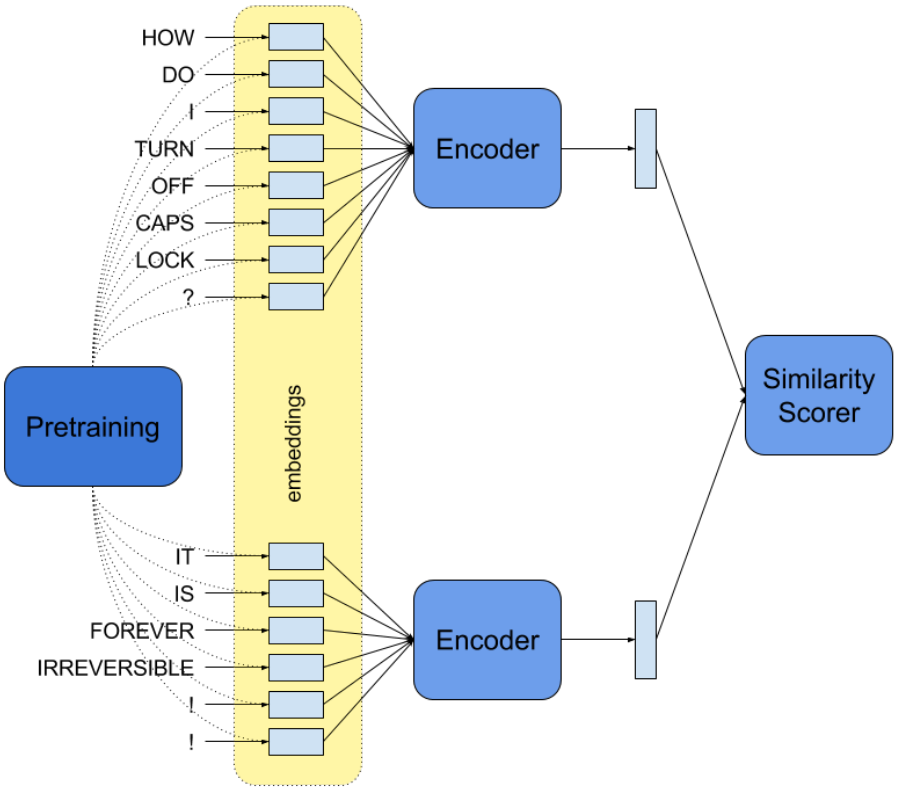

Figure 4.2: Illustration of a typical deep learning architecture for answer ranking. The question and the answer are represented as word embeddings that are optionally pretrained. Then an encoder is used to convert the embeddings into a fixed-sized vector. The same or two different encoders can be used for the question and the answer. Then a similarity scorer is used on the obtained vectors.

answer are usually represented as word embeddings (e.g. (Severyn and Moschitti, 2015a; Tang et al., 2015b; Tan et al., 2015; dos Santos et al., 2016)). Shen et al. (2015); dos Santos et al. (2016); Tymoshenko et al. (2016a) use the word2vec tool (the skip-gram and continuous bag-of-words models, described in Chapter 2) for pretraining; Amiri et al. (2016) use GloVe (Pennington et al., 2014), a count-based analogue of the word2vec models, that normalises and factorises the word co-occurrence matrix in order to obtain low-dimensional embeddings.

Encoder: The encoder transforms the word or character embeddings representations (not necessarily pretrained) of a text into a fixed-length vector. In factoid answer selection, convolutional neural networks were extremely popular (e.g.

Severyn and Moschitti (2015a); Tymoshenko et al. (2016a); Yu et al. (2014)), as CNNs are efficient and effective in encoding short texts. For non-factoid answer selection and answer selection in CQAs, LSTMs are often used to encode questions and answers (e.g. Tan et al. (2015); dos Santos et al. (2016)). Amiri et al. (2016) use a stacked denoising autoencoder (Vincent et al., 2010) to obtain representations of the questions and the answers. dos Santos et al. (2016) incorporate attention mechanisms into their encoder which could be either a recurrent or a convolutional neural network. This allows them to learn representations of text pairs that are aware of their similarity to one another. Shen et al. (2015) encode the question-answer pair into one vector simultaneously by first, calculating a word embedding pairwise similarity matrix for the question and the answer, and then, encoding this matrix with a convolutional neural network, similar to the one of Krizhevsky et al. (2012) designed for images. We could also mention the feature-based approaches here, as representing the text as a vector of handcrafted features is also a type of encoding.

Similarity Prediction: Several studies use cosine similarity on the representations of the question and the answer (e.g. dos Santos et al. (2016); Tan et al. (2015)). Another way to estimate the similarity or relevance between the question and the answer is to predict it using a feedforward neural network over their representations (Severyn and Moschitti, 2015a; Tymoshenko et al., 2016a; Nakov et al., 2016a). We will follow the latter approach, which will be described in Section 4.2.

Additional Features: Using an additional neural predictor over the representations instead of using the cosine similarity directly on the vector representations supports the incorporation of additional features. For instance, Severyn and Moschitti (2015a) add word overlap features, as well as calculate the similarity between the encodings. They address the task of factoid answer selection and use the TREC QA dataset (Wang et al., 2007), where the word overlap

is very high between the question and its correct answers (e.g. *Who founded the Black Panther organization?* and *The Black Panther party was founded by Seale and Huey Newton*). Other neural studies also use word overlap information when using this dataset (Amiri et al., 2016; Tymoshenko et al., 2016a). We use additional discourse features of Jansen et al. (2014); this approach is described in Chapter 7.2.

We present a comparative summary of the neural approaches to CQA-related tasks in Table 4.2. Please note that this is not a complete description of these studies, as most of these studies use other features and/or explore other approaches and/or tasks. For example, Amiri et al. (2016) and Lau and Baldwin (2016) evaluate their approaches on the tasks of word similarity and semantic textual similarity respectively. We only summarize the contributions related to answer selection and question similarity.

Some approaches span both the feature-based and the neural categories. For instance, Nakov et al. (2016a) take a feature-based approach that uses a feedforward neural network as a predictor. Tymoshenko et al. (2016a) combine tree kernels with convolutional neural networks. In Chapter 6.1 we also combine the neural approach with the approach based on discourse information. Bonadiman et al. (2017) use a multitask learning approach to jointly learn the answer ranking (both to the same and to related questions) and question-question similarity tasks. A CNN is used to encode the question and the answers, and then separate MLPs are used for prediction, depending on the task.

4.2 Methodology

We take a supervised learning to rank approach to answer reranking for community question answering. In the following sections we give a brief overview of learning to rank techniques and then introduce our approach to answer reranking.

Study	Task	Dataset	Pretraining	Encoder	Sim
Shen et al. (2015)	CQA AS	Baidu Zhidao	skip-gram	CNN over S-matrix (cosine sim between embeddings)	MLP
Bogdanova et al. (2015)	QD	Ask Ubuntu	skip-gram	CNN	cosine
Severyn and Moschitti (2015a)	Factoid AS	Trec QA	skip-gram	CNN + word overlap	MLP
dos Santos et al. (2015)	QD	Ask Ubuntu; English Stack Exchange	skip-gram	CNN; weighted BOW	cosine (combination)
Tan et al. (2015)	AS	TrecQA; Insurance QA	word2vec[?]	LSTM; LSTM+CNN with attention over the question	cosine
Tymoshenko et al. (2016a)	Factoid AS; AS	TrecQA; WikiQA	skip-gram	CNN (combined with SVM CTK using LR)	MLP
Bogdanova and Foster (2016)	CQA AS;	Yahoo! Answers	DBOW	Paragraph Vector model (Chapter 6)	MLP
dos Santos et al. (2016)	Factoid AS; NF AS;	Trec QA; WikiQA; InsuranceQA	skip-gram	Attentive pooling RNNs and CNNs	cosine
Lei et al. (2016)	QD	Ask Ubuntu	Encoder-decoder over similar questions; Denoising body into title.	RCNN, a variation of CNN with an additional gate to downweight certain bigrams;	cosine
Amiri et al. (2016)	QD; Factoid AS	Ask Ubuntu; Trec QA	GloVe; Layerwise pretraining	Stacked denoising autoencoder with context. Context is encoded from word matrix	cosine
Lau and Baldwin (2016)	QD	CQADupStack	skip-gram	Paragraph Vector Le and Mikolov (2014)	cosine
Bogdanova et al. (2017)	CQA AS;	Yahoo! Answers; Ask Ubuntu	-	GRU + discourse features	MLP

Table 4.2: Comparative Summary of Neural Approaches in CQA. Papers we (co)authored are in bold. **Abbreviations**: NF - non-factoid; AS - answer selection; QD - question duplication; *?* - not clear from the paper what model is used.

4.2.1 Learning to Rank Answers

Let $\boldsymbol{x}_i \in \mathbb{R}^d, i = 1, ..., m$ be a list of elements to be ranked. For example, given a query q, the elements are the documents that should be ranked according to their relevance to the query q. The number of elements m could be different for each query. Let $\boldsymbol{x}_i \triangleright \boldsymbol{x}_j$ denote that $\boldsymbol{x}_i$ is **preferred over** $\boldsymbol{x}_j$ and $\boldsymbol{x}_i \trianglerighteq \boldsymbol{x}_j$ denote that $\boldsymbol{x}_i$ is preferred over or equally preferred to $\boldsymbol{x}_j$. $\boldsymbol{x}_i \sim \boldsymbol{x}_j$ states that $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are equally preferred ($\triangleleft$ and $\trianglelefteq$ are defined similarly).

The binary relation of **preference** is usually assumed to define a **partial order** in $\mathbb{R}^d$, i.e. the following statements are true:

1. Reflexivity: $\forall \boldsymbol{a} : \boldsymbol{a} \trianglelefteq \boldsymbol{a}$
2. Transitivity: $\boldsymbol{a} \trianglelefteq \boldsymbol{b}$ and $\boldsymbol{b} \trianglelefteq \boldsymbol{c} \Rightarrow \boldsymbol{a} \trianglelefteq \boldsymbol{c}$
3. Antisymmetry: $\boldsymbol{a} \trianglelefteq \boldsymbol{b}$ and $\boldsymbol{b} \trianglelefteq \boldsymbol{a} \Rightarrow \boldsymbol{a} \sim \boldsymbol{b}$

Usually, the ranking task assumes the relation to define a **total order**: the preference relation is defined for all possible pairs, i.e. all elements are **comparable** to each other in terms of preference:

4. Totality: $\forall \boldsymbol{a}, \boldsymbol{b} : \boldsymbol{a} \trianglelefteq \boldsymbol{b}$ or $\boldsymbol{b} \trianglelefteq \boldsymbol{a}$

Learning-to-rank methods are supervised machine learning methods that learn a ranking model given a training set, for which the ranking is known, i.e. the preference relation is defined for all possible pairs. Ranking models usually assume that there exists a **ranking function** $f : \mathbb{R}^d \to \mathbb{R}$, such that $f(\boldsymbol{a}) > f(\boldsymbol{b}) \iff \boldsymbol{a} \triangleright \boldsymbol{b}$.

Supervised learning to rank methods are usually trained by minimising a loss function L on the training set. There are three main approaches to learning to rank depending on the type of loss function they use:

pointwise ranking methods usually consider each $\boldsymbol{x}_i$ independently, and transform the ranking problem into either a regression or a classification task. The pointwise loss function treats every instance independently: $L : \mathbb{R}^d \to \mathbb{R}$

pairwise methods consider pairs of instances and transform the task into a pairwise classification, with the loss function defined for each pair of instances: $L : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

listwise methods represent a more straightforward approach to learning to rank, as they take the list of instances as an input and predict a permutation, i.e. the ranking of those instances. The listwise loss function is defined for any number of instances: $L : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \times ... \times \mathbb{R}^d \to \mathbb{R}$ In practice, representing listwise loss functions is challenging. Cao et al. (2007) suggest to represent the output of the ranking as a probability distribution, and thus, the listwise loss can be defined as any metric on two probability distributions, e.g.cross-entropy.

Let us reformulate the problem of ranking for the task of answer ranking. For each question q asked on a user forum or a CQA, let $[\boldsymbol{x}_i \in \mathbb{R}^d, i = 1, ..., m]$ be a list of its user-provided answers. The number of answers m could differ for each question. We assume that for each question q there exists the best answer, i.e. $\exists k : \boldsymbol{x}_k \triangleright \boldsymbol{x}_i \ \forall i \in [1, m], i \neq k$. In reality, some questions asked online have no answers of desirable quality, however, in the datasets we use this property always holds.

In contrast to the traditional learning to rank setting, in answer ranking the preference binary relation is not always defined for all answer pairs, i.e. the relation defines only a partial order, not a total order. However, in those cases where not only the information about the best answer but the scores for all the answers are available, the relation of the total order is defined. The number of answers to be ranked could vary from two to dozens or possibly hundreds. This makes training the listwise approaches rather cumbersome. We focus on the pointwise learning to rank methods, as the order is not always defined for all possible pairs of answers. The pointwise learning to rank approach casts the ranking problem as a classification or regression task. In the next section we describe our approach more in detail.

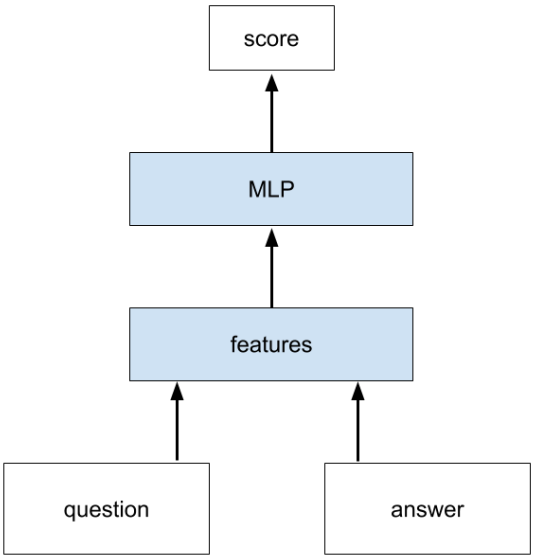

Figure 4.3: Our general approach to answer ranking. The blue boxes are components that can learn from data. The features component can, for instance, be an RNN encoder that is trained together with the MLP.

4.2.2 Answer Ranking with Multilayer Perceptron

In most of our experiments we use a simple fully-connected feedforward neural network, i.e. a multilayer perceptron described in Chapter 2.3.1, to predict the best answer. As we choose to take the pointwise learning to rank approach, given a question, we consider each of its answers independently and use the MLP to predict the probability of the answer being correct. As shown in Figure 4.3, the question and the answer are first represented as features. Here we use the word *features* in a broad sense of a vector representation, meaning that these features could not only be extracted but learnt. For instance, one way to obtain them is to run a recurrent neural network on the question-answer pair. Then these features are passed as the input to the MLP, which predicts a score, i.e. the probability of this answer being correct for this question. If a neural network, such as an RNN, is used to learn the features, it is trained together with the MLP. The training is done by minimising the cross-entropy on the training set.

4.2.3 Data Representation

Our approach requires the question-answer pairs to be represented as a fixed-size vector that is received by an MLP. We consider several different ways to represent the question-answer pair as a vector:

Paragraph Vector: We learn representations for the question and the answer using the Paragraph Vector model (Le and Mikolov, 2014), a simple unsupervised technique for learning distributed representations of documents. See Chapter 5 for details.

Recurrent Neural Networks: These networks described in Section 2.3.3 can encode a variable length text as a fixed-length vector. We usually run separate RNNs for the question and the answer and then concatenate the obtained encodings. The MLP and the RNNs are trained together. See Chapter 6 for details.

Convolutional Neural Networks: Just like the RNNs, this type of networks can be used to encode a variable length text as a fixed-length vector. We also use separate CNNs for the question and the answer, concatenate them and pass the obtained representation to the MLP. See Chapter 6 for details.

Combination of Recurrent and Convolutional Networks We first encode the question with a forward and a backward RNN and then use the outputs of the RNNs and the word embeddings as three input channels of a CNN, which encodes the question. The answer is encoded in the same way. The rest of the configurations is the same as when only an RNN or only a CNN is used for encoding. See Chapter 6 for details.

Discourse Features: Jansen et al. (2014) describe a system where discourse-related features were combined with distributed representations of words obtained with the skip-gram model (Mikolov et al., 2013a). This system performed well in the task of answer reranking. See Chapter 7 for details.

4.3 Resources

Community question answering websites provide us the labelled data we need for training our models. In particular, we assume that the answer selected by the users as the best one is the true gold label, and the rest are negative examples.[3] An example of a question posted on Yahoo! Answers (YA) website is shown in Figure 4.1.

In our experiments, we use two datasets from different CQAs. For comparability with previous research, we use the dataset created by Jansen et al. (2014). It contains 10K *how* questions from Yahoo! Answers. 50% of it is used for training, 25% for development and 25% for testing. This dataset was sampled from a corpus of *how* questions initially created by Surdeanu et al. (2011). The original dataset was created in two steps:

1. They selected questions that have the best answer chosen and match the following regular expression:

   `how (to|do|did|does|can|would|could|should).`

2. Questions and answers with fewer than five words were removed. This heuristic is supposed to remove questions and answers of low quality, e.g. *How to be great?* and *I don't know.*

The 10K dataset contains only questions that had at least four user-generated answers. Yahoo! Answers allows users to select a category when posting a question. The questions in the YA dataset contain the category information. There are 283 distinct categories present in the training set, including *South Africa*, *Allergies* and *Non-Alcoholic Drinks*. We only use this information in error analysis in Section 7.4. Some examples from this dataset can be found in Table 4.3.

To evaluate our approach on a more technical domain, we create a dataset of questions from the Ask Ubuntu (AU)[4] community. The dataset contains 13K ques-

[3] This assumption is not always true. We will discuss this in Chapter 7.
[4] `http://askubuntu.com/`

Question: how do you cut onions without crying?
Gold: Use a sharp knife because if the onions are cut cleanly instead of slightly torn (because of a dull knife) they will release less of the chemical that makes you cry. Lighting a candle also helps with this, (...) I hope this helps.
Other Answers: - Watch a comedy. - Put onion in the chop blender - close ur eyes... - Sprinkle the surrounding area with lemon juice. - Choose one of the followings after cutting the head and tail of the onion, split in half and peel off the skin. 1. Keep on chopping with your knife 2. Cut in quarters and put in choppers.

Table 4.3: Example question and answers from the Yahoo! Answers dataset.

tions, of which 10K are used for training, 0.5K for development and 2.5K for testing. We create the AU dataset in the same way as the YA dataset was created: for each question, we only rank the answers provided in response to this question, the answer labelled as the best by the question's author is considered the correct answer. We make sure that the dataset contains only questions that have at least three user-provided answers, have the best answer selected, and this answer has a non-negative score. Example questions from this dataset can be found in Table 4.4

Question: Can't shutdown through terminal. When ever i use the following `sudo shutdown now; sudo reboot; sudo shutdown -h` my laptop goes on halt (...) is there something wrong with my installation?
Gold: Try the following code `sudo shutdown -P now` (...) -P Requests that the system be powered off after it has been brought down. -c Cancels a running shutdown. -k Only send out the warning messages and disable logins, do not actually bring the system down.
Other Answers: - Try `sudo shutdown -h now` command to shutdown quickly. - Try `init 0` init process shutdown all of the spawned processes/daemons as written in the init files

Table 4.4: Example question and answers from the Ask Ubuntu dataset.

The datasets have significant differences – see Table 4.5 for more information. While the Yahoo! Answers dataset has very short questions (10.8 on average) and relatively long answers (50.5 words), the Ask Ubuntu questions can be very long,

Dataset	Avg question title length	Avg question body length	Avg answer length	Vocabulary size
Yahoo! Answers	10.9	-	50.5	63.6K
Ask Ubuntu	8.74	112.14	95.04	144.9K

Table 4.5: Comparative statistics on the datasets used in the answer reranking experiments.

as they describe non-trivial problems rather than just ask questions. The average length of the Ask Ubuntu questions is 112.14 words, with the average answer being about 95 words.

4.4 Experimental Setup

4.4.1 Baselines

Following Jansen et al. (2014) and Fried et al. (2015), we implement three baselines:

Random Baseline: this baseline selects the best answer randomly.

Candidate Retrieval Baseline (CR): this system uses the same scoring as in Jansen et al. (2014): (1) the questions and the candidate answers are represented using *tf-idf* (Salton and McGill, 1986) over lemmas; (2) the candidate answers are ranked according to their cosine similarity to the respective question.

Chronological Baseline: This baseline ranks answers using the date and time when they were originally posted, i.e. selects the first posted answer as the best. The YA dataset does not have information on when the answers were posted, thus, we only evaluate this approach on the AU dataset.

On the YA dataset, we also compare our results to the ones reported previously on the same dataset:

Jansen et al. (2014) describe answer reranking experiments on the YA dataset using a diverse range of features incorporating syntax, lexical semantics and

discourse. In particular, they show how discourse information (obtained either via a discourse parser or using shallow techniques based on discourse markers) can complement distributed lexical semantic information.

Fried et al. (2015) improve on the lexical semantic models of Jansen et al. (2014) by exploiting indirect associations between words using higher-order models.

4.4.2 Evaluation Metrics

We follow Jansen et al. (2014) and Fried et al. (2015) and in most experiments we evaluate our reranking systems using two information retrieval metrics:

Precision-at-1 (P@1): In information retrieval, precision at n ($P@n$) is the proportion of relevant documents retrieved among the top n results. We always have only one relevant document, i.e. the best answer, for each query, i.e. question. The best answer should be ranked first, that is why we use P@1 as the main metric. The P@1 in case of answer reranking is the proportion of best answers ranked first, i.e.

$$P@1 = \frac{best\_is\_first}{|Q|} \tag{4.6}$$

where $best\_is\_first$ is the number of best answers ranked first, and $|Q|$ is the total number of questions. This metric could take any real value between 0 and 1, with 1 being the perfect score.

Mean Reciprocal Rank (MRR): The reciprocal rank (RR) is the reciprocal of the position at which the first relevant document was retrieved. For example, if the first relevant document is ranked first, the RR is 1. If it appears at the third position (with the first and the second being irrelevant), the RR is $\frac{1}{3}$. MRR is the RR averaged for all queries. MRR is usually used when there is only one relevant document, which is exactly the case of answer reranking.

We calculate the MRR as follows:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{best\_ranked\_as} \tag{4.7}$$

where *best_ranked_as* is the rank of the best answer and $|Q|$ is the number of questions. As we have only one relevant answer for each question, MRR becomes equivalent to **Mean Average Precision (MAP)**.

We will report both the P@1 and the MRR in percent (0-100) to make the results better readable and avoid confusion when comparing our results to the previous work that reported these measures in this way (Jansen et al., 2014; Fried et al., 2015).

We test statistical significance with one-tailed bootstrap resampling with 10,000 iterations as in Graham et al. (2014).

4.4.3 Data Preprocessing

For the AU dataset, we keep the code the posts contain within a `code` tag. For both the YA and the AU datasets, we only perform very shallow preprocessing: we tokenise the texts with the tokeniser which is packaged with the Stanford parser[5]. We lowercase the tokenised data and exclude words that occur five times or fewer in the training set. This results in 14829 and 37530 distinct words for the YA dataset and the AU datasets respectively. All other words are mapped to an `<UNK>` tag.

4.5 Summary

This chapter introduced answer ranking in Community Question Answering websites, the task that we will deal with throughout the following three chapters. We presented an overview of the previous work on the task and related tasks, roughly dividing the approaches into feature-based ones, i.e. the ones where the

[5]we used version 3.6.0 `http://nlp.stanford.edu/software/stanford-parser-full-2015-12-09.zip`

ranking/classification performance is achieved due to the quality of the features representing question-answer pairs; and neural approaches that achieve the ranking performance due to the representation capacity of the neural architecture. We described our approach to the problem, which is a pointwise learning to rank approach, where we use a neural system with a multilayer perceptron serving as the final predictor. We introduced the two datasets we use in most experiments: the dataset of *how* questions from Yahoo! Answers and the dataset of Ask Ubuntu questions. We also presented our experimental setup and the evaluation metrics.

Chapter 5

Answer Ranking with Paragraph Vector

Most previous approaches to non-factoid answer ranking were feature-based. In this chapter we explore one of the most straightforward ways to avoid feature engineering. In particular, we use the Paragraph Vector models (Le and Mikolov, 2014) to obtain question and answer representations in an unsupervised manner from a large unlabelled corpus. We then use these representations in combination with a multilayer perceptron, as described in Chapter 4. Only the parameters of the multilayer perceptron are updated during training, while the representations learnt with Paragraph Vector model remain the same.

Paragraph Vector (PV) model (Le and Mikolov, 2014) is a model for learning distributed representations for documents (the document can be a sentence, a paragraph or a piece of text of an arbitrary length, we will refer to it as a document in this section). The PV is an extension of the skip-gram and the continuous bag-of-words models that we described in Chapter 2.5. It simply treats the document as if it were another token shared across all the word windows in it, and thus can learn a vector representation for it using the CBOW and the skip-gram models. The PV consists of two models: a Distributed Memory (DM) model and a Distributed Bag-of-Words (DBOW) model. The DM and the DBOW models differ in the way they train

these representations. The DM model trains them similarly to the CBOW model with the following difference: for every word window, when predicting the word, the document representation is concatenated or averaged with all the word vectors. The document vector stays the same for all the word windows. The DBOW model is trained similarly to the skip-gram model: given a document vector, the DBOW model is trained to predict the words in this document. In both PV models the word vectors are shared across the documents, just like in the original CBOW and skip-gram models. In the next paragraphs we explain the differences between these models and the word embedding models presented in Chapter 2.5 more formally.

This chapter is structured as follows: first we describe the distributed memory (DM) model in Section 5.1.1 and the distributed bag-of-words (DBOW) model in Section 5.1.2. We present answer ranking experiments where question and answer representations are obtained with the Paragraph Vector model in Section 5.2. In particular, Section 5.2.2 compares the performance of the DBOW and the DM models. Section 5.2.3 investigates the impact of the representation dimensionality on the answer ranking performance. Section 5.2.4 evaluates two settings: when the questions and answers were obtained by including them in the pretraining corpus versus when their representations were inferred using a trained model. In Section 5.2.5 we experiment with inferring the vectors using an in-domain versus an out-of-domain corpora. Finally, we summarise the answer ranking results obtained with the Paragraph Vector representations in Section 5.3.

5.1 Paragraph Vector

5.1.1 Distributed Memory Model

Just as every word is represented by a row in a word embedding matrix $\mathbf{W}$ of size $V \times d$, where V is the vocabulary size and d is the desired dimensionality of the embeddings (see Chapter 2.5.1), every document is represented by a row in a matrix $\boldsymbol{D}$ of size $N \times d$, where N is the number of documents in the training corpus. We

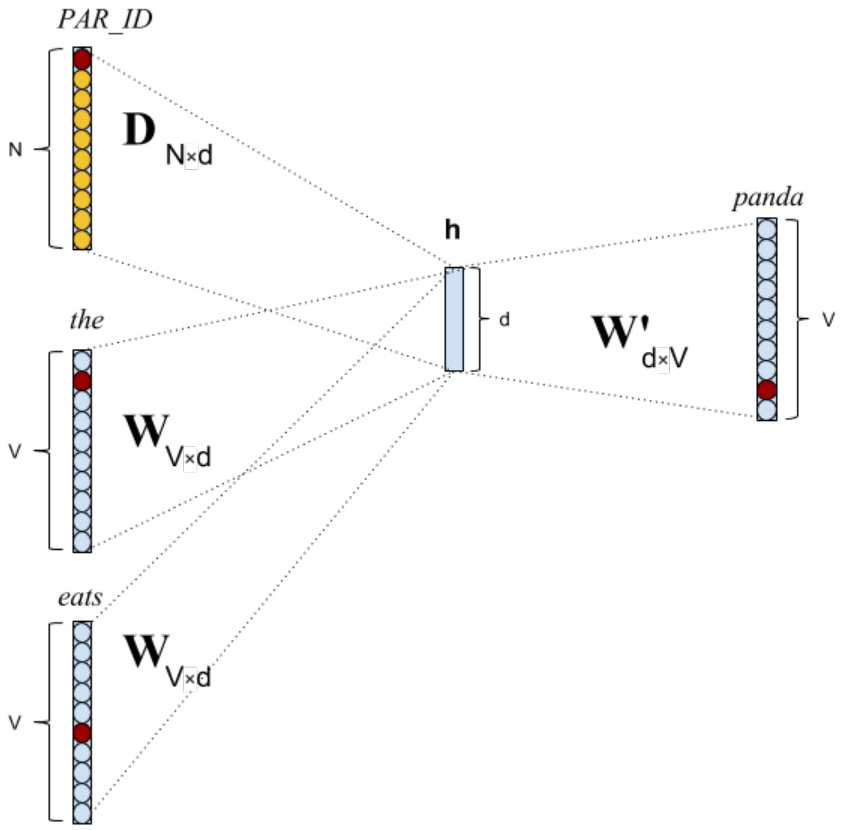

Figure 5.1: Illustration of the DM model for the word *panda* in context *the panda eats*.

illustrate the DM model in Figure 5.1. The only difference from the CBOW model described in Chapter 2.5.1 is that the vector $\boldsymbol{h}$ is obtained not only by averaging (or concatenating) the word vectors but also a document vector, i.e:

$$\boldsymbol{h} = \frac{1}{C+1}(\boldsymbol{v}_{w_1} + ... + \boldsymbol{v}_{w_C} + \boldsymbol{v}_d) \tag{5.1}$$

where $\boldsymbol{v}_{w_i}$ is the embedding of the i-th context word and $\boldsymbol{v}_d$ is the document vector. The document vector is shared across all word windows in the document. Consider an example document that consists of only the sentence *The panda eats shoots and leaves* and let us assume its id to be *PAR_12*. Let us also assume the word window to be equal to one, i.e. we consider one word to the left and one word to the right. The DM's training instances will include:

input=[*the*, *eats*, *PAR_12*], label=*panda*

input=[*panda*, *shoots*, *PAR_12*], label=*eats*

input=[*eats*, *and*, *PAR_12*], label=*shoots*

input=[*shoots*, *leaves*, *PAR_12*], label=*and*

5.1.2 Distributed Bag-of-Words

This model is similar to the skip-gram model for learning word embeddings. Given a document vector as an input, the model learns to predict words randomly sampled from this document. Figure 5.2 illustrates the DBOW model for the example from the last section. The only difference from the skip-gram model is that the vector $\boldsymbol{h}$ is now the embedding of the input document:

$$\boldsymbol{h} = \boldsymbol{D}^{\top}\boldsymbol{d_i} \tag{5.2}$$

where $\boldsymbol{d_i}$ is a one-hot encoding of the document's id.

Given the example from the above paragraph, the DBOW's training instances will include:

input=[*PAR_12*], label=*panda*

input=[*PAR_12*], label=*the*

input=[*PAR_12*], label=*shoots*

input=[*PAR_12*], label=*and*

input=[*PAR_12*], label=*leaves*

Both the DM and the DBOW models are trained in the same way as the CBOW and the skip-gram models, e.g. using negative sampling or the hierarchical softmax (see Section 2.5.1 for details).

In the paper that originally presented the PV model (Le and Mikolov, 2014), the DBOW model is reported to be inferior to the DM model, and the authors encourage the use of DM or its combination, i.e. concatenation, with the DBOW representations.

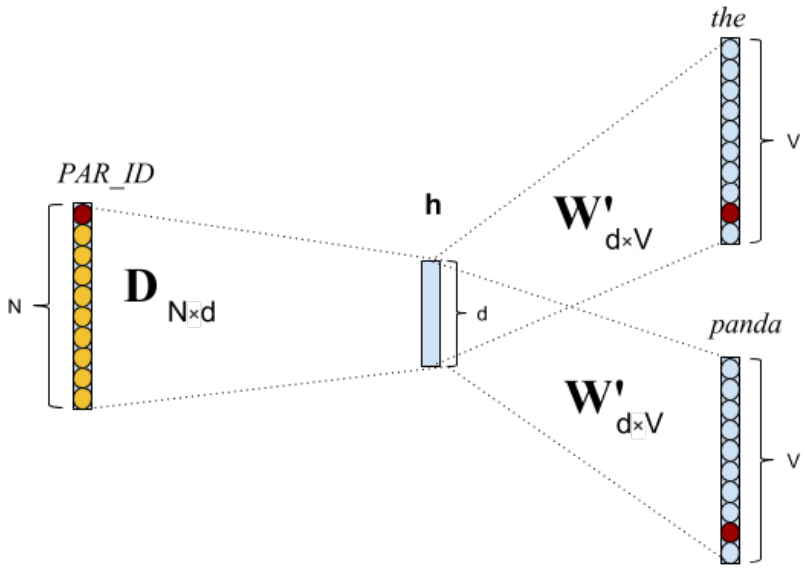

Figure 5.2: Illustration of the DBOW model for learning a vector representation of the sentence *The panda eats*

5.2 Experiments

To obtain the representations for the YA questions and answers, we train the PV models on the questions[1] from the *L6 Yahoo! Answers Comprehensive Questions and Answers* corpus obtained via Webscope.[2] This dataset contains about 4.5M questions from Yahoo! Answers along with their user-generated answers, and was provided as training data at the recent TREC LiveQA competition (Agichtein et al., 2015), the goal of which was to answer open-domain questions coming from real users in real time.[3] The YA dataset prepared by Jansen et al. (2014) and described in Section 4.3, was initially sampled from this larger dataset. The YA dataset was added to the L6 corpus before the training.

To obtain the representations for the AU dataset, we train the PV on the January

[1]The gensim implementation of doc2vec available at the time we conducted the experiments required that all the vector representations were stored in RAM, and we did not have a machine with enough RAM available, that is why the corpus was enriched with 4.5M questions only. This was improved in later versions of gensim.

[2]`http://webscope.sandbox.yahoo.com/`

[3]`https://sites.google.com/site/trecliveqa2015/`

Corpus	# of documents	# of tokens	Vocabulary size
L6 Yahoo! Answers	4.5M	60M	1M
Ask Ubuntu dump	1.4M	97M	183K

Table 5.1: Details on the corpora used to train the Paragraph Vector models.

2015 Ask Ubuntu dump[4], from which the AU dataset was sampled. In most of our experiments the test set is also included in the pretraining corpus, in Section 5.2.4 we also explore how the model performs when the test set is not available at the pretraining time, and the test representations are inferred. We want to emphasise that the L6 and the large AU datasets are only used for unsupervised pretraining – no meta-information is used in our experiments. We report the statistics about the two corpora in Table 5.1.

The PV models are trained for twenty epochs with an initial learning rate of 0.025, at each epoch the learning rate was decreased by 0.001. We use the window size of three (i.e. three words to the left, and three words to the right). The models are trained with negative sampling using ten contrastive examples. The mean of the context word representations is used in training the DM model. We use the gensim (Řehůřek and Sojka, 2010) implementation of PV also know as *doc2vec* to train the models.[5]

The question and the answer are represented as separate documents, and their representations are obtained during pretraining and are then concatenated before being passed to a multilayer perceptron that predicts the score for the answer. As shown in Figure 5.3, the first layer transforms question-answer pairs into their PV representations, i.e. the vector representation for a question-answer pair (q, a) is a concatenation of the distributed representations q and a for the question and the answer respectively. Each representation is a real-valued vector of a fixed dimensionality d, which is a parameter to be tuned. The MLP is trained with SGD. We only update the weights and biases of the MLP during the training, the PV representations remain fixed. We use early stopping on the development set. We regularise

[4] `https://archive.org/download/stackexchange/askubuntu.com.7z`
[5] `https://radimrehurek.com/gensim/models/doc2vec.html`

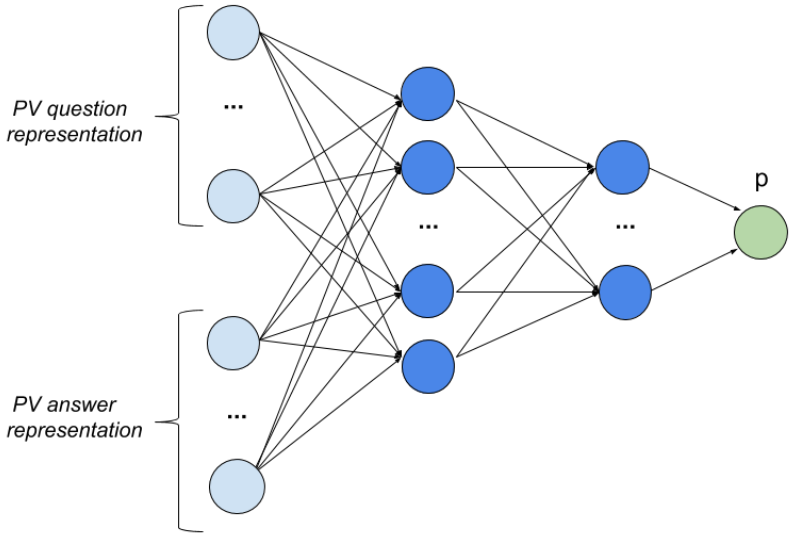

Figure 5.3: Illustration of the method based on the Paragraph Vector and an MLP.

the MLP with L2 weight decay with a 10^{-6} regularisation rate.

5.2.1 Ask Ubuntu Question Representation

Recall that in the AU dataset the questions have a title which usually briefly summarises the problem and a body that explains the problem in detail. In order to decide if we use the title, the body or the concatenation of both as the input to MLP, we evaluate the approach with the three possible representations on the development set: (1) representing the question as its title (2) representing the question as its body and (3) concatenating the two representations. Table 5.2 reports the development performance of the 200-dimensional DBOW model with an MLP with 256, 128, 64, 32 and 16 hidden units. The results suggest that using the concatenation of the title and the body of the question as its representation leads to a better performance in terms of P@1 than when only the title or only the body is used. However, using the body only provides better results than when using the title only. In the following experiments we will use the concatenation of the title and the body

Question Representation	dev P@1	dev MRR
title	42.40	64.34
body	43.20	**64.97**
title + body	**44.00**	64.88

Table 5.2: Development performance of the 200-dimensional DBOW model on the AU dataset with different question representations; *title* means that only question title was used, *body* means only question body was used; *title+body* means that a concatenation of both the title and the body was used. An MLP with 256, 128, 64, 32 and 16 hidden units was used.

as the representation of the Ask Ubuntu questions.

5.2.2 DBOW versus DM

We first evaluate the DBOW model versus the DM model on the task of answer ranking, i.e. the vectors for the questions and the answers are obtained by pre-training one of these models on a large corpus, and these vectors are passed to an MLP. We evaluate representations with 100 and 200 dimensions. Note, that if the dimensionality of the PV representations is 100, the input layer of the MLP has 200 dimensions, as we concatenate the question and the answer vectors before passing them to the network.[6] As the original Paragraph Vector paper (Le and Mikolov, 2014) suggests, we also evaluate the concatenation of the representations obtained by separately training the DBOW and the DM models. We refer to this approach as DBOW-DM in Table 5.3.

We select the best parameters for the DM, the DBOW and the DBOW-DM models on the development set, and apply these models to the test set. For comparison with recent work in answer ranking (Jansen et al., 2014; Sharp et al., 2015), we also evaluate the averaged word embedding vectors obtained with the skip-gram model (Mikolov et al., 2013c) (henceforth referred to as the *SkipAvg* model). Table 5.3 presents the experimental results. On both datasets, the distributed representations, including the SkipAvg model, beat both random and candidate retrieval

[6]or 300 for the AU dataset, if the body and the title of a question and represented as separate vectors. We experiment with this in Section 5.2.1

baselines by a large margin. On the AU dataset, the SkipAvg model performs at the same level as the chronological baseline. On the YA dataset, its performance falls in between the models of Jansen et al. (2014) and Fried et al. (2015). Paragraph Vector representations clearly outperform the averaged word representations on both datasets. Both paragraph vector models – DBOW and DM – provide similarly high performance, however the DBOW model provides a slightly better P@1 on both datasets. We gain an additional small improvement over the DBOW performance by concatenating DBOW and DM representations.

Yahoo! Answers		
Model	**P@1**	**MRR**
DBOW-DM	**37.37***	**57.05**
DBOW	37.01*	56.88
DM	35.61*	55.58
SkipAvg	31.25	52.56
Random baseline	15.74	37.40
CR baseline	22.63	47.17
Jansen et al. (2014)	30.49	51.89
Fried et al. (2015)	33.01	53.96
Ask Ubuntu		
Model	**P@1**	**MRR**
DBOW-DM	**41.44***	**64.37**
DBOW	41.24*†	63.96
DM	41.12*	63.98
SkipAvg	37.68	61.90
Random baseline	26.60	53.64
CR baseline	35.36	60.17
Chronological baseline	37.68	60.06

Table 5.3: Results of the PV-MLP system on the AU dataset. *The improvements over the baselines are statistically significant with $p < 0.05$. The improvements of the DBOW over the DM model on the YA dataset is statistically significant ($p < 0.05$); †The improvement of the DBOW over the DM on the AU dataset is not statistically significant ($p > 0.05$). All significance tests are performed using one-tailed bootstrap resampling with 10,000 iterations.

The DM is reported to be superior to the DBOW in the original Paragraph Vector paper by Le and Mikolov (2014), as the DM can account for the word order

and the DBOW cannot. Our experiments, on the contrary, show that the DBOW model outperforms the DM model on the YA dataset and performs at the same level on the AU data. As the evaluation of the PV models by Lau and Baldwin (2016) suggests, the DBOW might perform better than the DM because the latter needs much longer to converge. Lau and Baldwin (2016) report that whilst the optimal[7] number of training epochs for the DBOW was 20, the DM model needed to be trained for 600 epochs to achieve its optimal performance on the task of duplicate question detection. In our experiments we trained both models for twenty epochs. Training the DM model significantly longer might boost its performance.

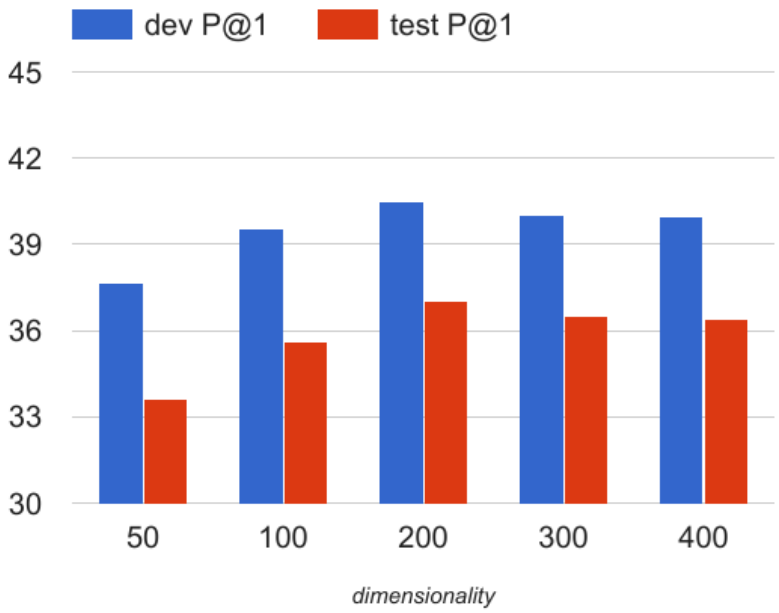

Figure 5.4: Development P@1 and test P@1 for the DBOW model with 50, 100, 200, 300 and 400-dimensional representations on the YA dataset.

5.2.3 The Impact of the Paragraph Vector Size

We trained several DBOW models with the same parameters except for the dimensionality of the representations. The optimal dimensionality of the representations usually depends on the task and the amount of training data. Figure 5.4 reports

[7] on the task of duplicate question detection

the best development P@1 and the test P@1 on the YA dataset for the models with 50, 100, 200, 300 and 400 dimensions. As Figure 5.4 shows, the performance increases when increasing the representation dimensionality from 50 to 100 and from 100 to 200 dimensions. However, increasing the dimensionality beyond 200 seems to decrease the performance slightly.

Yahoo! Answers			
Model	**Representations**	**P@1**	**MRR**
DBOW	extracted	**37.01**†	**56.88**
DBOW	inferred	36.45	56.13
DM	extracted	35.61*	55.58
DM	inferred	34.53	54.36
Jansen et al. (2014)		30.49	51.89
Fried et al. (2015)		33.01	53.96
Random baseline		15.74	37.40
CR baseline		22.63	47.17
Ask Ubuntu			
Model	**Representations**	**P@1**	**MRR**
DBOW	extracted	41.24	63.96
DBOW	inferred	**41.48**†	**64.33**
DM	extracted	41.12*	63.98
DM	inferred	38.88	62.70
Random baseline		26.60	53.64
CR baseline		35.36	60.17
Chronological baseline		37.68	60.06

Table 5.4: Comparison of the MLP performance using the extracted PV representations versus using the inferred PV representations. *The improvements of the DM model with extracted vectors over the DM model with inferred vectors is statistically significant ($p < 0.05$). †The improvements of the DBOW model are not statistically significant ($p > 0.05$).

5.2.4 Paragraph Vector Representations for New Documents

In the experiments reported above, we obtained the PV representations for questions and answers by including them in the corpus that was used to train the PV models. The representations for the development and the test sets were obtained in the

same way. The same has been done in the original PV paper (Le and Mikolov, 2014). However, in a real-world case scenario, including the test documents in the pretraining corpus might not be always possible, as it may not be available at that stage, e.g. when performing online question answering. An alternative to this approach is to infer the vector representation for a document using a trained PV model. This is done using gradient descent, similarly to the training phase described in Section 5.1.1 for the DM model and in Section 5.1.2 for the DBOW model, with the only difference being that the weights of the model and the word vectors remain fixed, i.e. only the document vector is updated. For every word in the new document, the model is trained to predict this word given the document vector. Only the document vector is updated during the training, the word vectors and the weights of the model remain fixed. Any words that were not present during the original training are ignored.

In order to see how well the PV models perform in the answer ranking task when the dataset is not available at pretraining time, we use the same models to infer vectors for all the instances in our dataset, including the training set. We infer the vectors for 500 iterations with an initial learning rate of 0.01 and final learning rate of 0.0001, as these settings were found optimal by Lau and Baldwin (2016) for the tasks of duplicate question detection and text semantic similarity. Table 5.4 reports the results for the YA and AU datasets respectively.

The results suggest that the inferred DBOW representations provide comparable (YA dataset) or even slightly better (AU dataset) performance on the task of answer ranking versus when the data is included in the pretraining corpus (*extracted*). However, these differences are not statistically significant. For the DM model, the inferred representations are significantly inferior to the extracted ones, although these representations still beat all the baselines.

5.2.5 The Impact of the Pretraining Corpus

In order to compare the performance of the model when pretrained on in-domain versus out-of-domain data, we infer the vectors for the two datasets using a DBOW model trained by Lau and Baldwin (2016). This model was trained on the English Wikipedia: each paragraph was treated as a separate document, resulting in 32M documents. We compare the answer ranking performance of the models where the vectors were inferred using in-domain (L6 Yahoo! Answers for the YA dataset, and Ask Ubuntu data dump for the AU dataset) versus the ones that used the vectors inferred using the model trained on Wikipedia.

Yahoo! Answers			
Model	**Corpus**	**P@1**	**MRR**
DBOW	in-domain	**36.45**†	**56.13**
DBOW	Wikipedia	35.61	56.05
Jansen et al. (2014)		30.49	51.89
Fried et al. (2015)		33.01	53.96
Random baseline		15.74	37.40
CR baseline		22.63	47.17
Ask Ubuntu			
Model	**Corpus**	**P@1**	**MRR**
DBOW	in-domain	**41.48***	**64.33**
DBOW	Wikipedia	40.20	63.57
Random baseline		26.60	53.64
CR baseline		35.36	60.17
Chronological baseline		37.68	60.06

Table 5.5: Comparison of the MLP performance using the DBOW representations inferred using a model trained on in-domain data versus the one trained on Wikipedia. †On the YA dataset, the improvement is not statistically significant ($p > 0.05$). *On the AU dataset, the improvement is statistically significant ($p < 0.05$).

Table 5.5 reports the results. The representations inferred using the model pretrained on an in-domain corpus provide better answer ranking results for both datasets. On the YA dataset, the improvement of the model pretrained on in-domain data over the model pretrained on Wikipedia is not significant, however, it is significant on the AU dataset. This indicates that for a highly technical domain,

like the one of the AU dataset, it is more important to have in-domain data for pretraining. Even though the performance of the models that use vectors inferred with the model trained on Wikipedia is not as high as when in-domain data is used, they still outperform all the baselines, even for the AU dataset, which is highly technical. This result suggests that a pretrained general purpose Paragraph Vector model could be used to infer vectors for answer ranking.

5.3 Summary

Our general approach to answer ranking requires vector representations of question-answer pairs. In this chapter we used general purpose distributed document representations provided by Paragraph Vector models to represent question-answer pairs. The main findings of our experiments are:

- representing the question-answer pair with Paragraph Vector model is clearly superior to the use of averaged word vectors;
- the use of the DBOW model is more favourable than the DM model in the task of answer ranking, especially when inferring the representations using a pretrained model;
- a smaller amount of unlabelled data taken from a similar source as the dataset is more useful for training representations than a larger out-of-domain set.

In the experiments reported in this chapter we did not perform an extensive hyperparameter search. Although the results could potentially be improved by better hyperparameter tuning, it is clear that the Paragraph Vector provides document representations suitable for the task of answer ranking.

Chapter 6

Learning Representations for Answer Ranking

In the last chapter we performed answer ranking using the pretrained general purpose Paragraph Vector representations of Le and Mikolov (2014) for questions and answers. In this chapter instead of using pretrained representations, we learn them together with the task itself. To do this, we use two encoder networks that are trained together with the final MLP predictor. In contrast to the approach based on the Paragraph Vector model, learning the representations together with the ranking does not require pretraining and allows us to learn from the training set only. However, the models we describe here can be pretrained, and we will explore this in Chapter 7.

We use recurrent and convolutional neural networks (described in Chapter 2) as the encoder, i.e. the network that converts an object, which in our case is, a question or an answer, into a fixed-length vector. We compare the answer ranking performance of two widely used RNN architectures, the Long Short Term Memory networks (Hochreiter and Schmidhuber, 1997) and the Gated Recurrent Networks (Cho et al., 2014b) (usually abbreviated as GRU for Gated Recurrent Unit). We also compare the recurrent neural networks with the convolutional networks for the purposes of encoding questions and answers for answer ranking and propose a

novel architecture that combines the benefits of the two types of encoders.

This chapter is structured as follows: Section 6.1 introduces an approach to answer ranking that uses recurrent neural networks to encode questions and answers. Section 6.2 explores an approach where convolutional neural networks are used instead of recurrent ones. In Section 6.3 we describe our Multi-Channel Recurrent Convolutional Neural Network (MC-RCNN), a novel architecture for text encoding, that combines the recurrent and the convolutional architectures, and evaluate it on the task of answer ranking. Finally, we summarise the results and draw conclusions in Section 6.4.

6.1 RNN Encoder for Answer Ranking

We follow Bahdanau et al. (2014) and Cho et al. (2014b), and use a bidirectional[1] RNN as an **encoder**, i.e. a network that learns fixed-length vector representations of objects. Given a question-answer pair, we use two separate RNNs with either an LSTM or a GRU cell to encode the question and the answer. Let $(\boldsymbol{w}_1^q, \boldsymbol{w}_2^q, ..., \boldsymbol{w}_k^q)$ be the sequence of question word embeddings and $(\boldsymbol{w}_1^a, \boldsymbol{w}_2^a, ..., \boldsymbol{w}_p^a)$ be the sequence of answer word embeddings. The first RNN encodes the sequence of question words into the sequence of context vectors $(\boldsymbol{h}_1^q, \boldsymbol{h}_2^q, ..., \boldsymbol{h}_k^q)$, i.e.

$$f_{RNN}^q(\boldsymbol{w}_i^q, \boldsymbol{\theta}_q) = \boldsymbol{h}_i^q \tag{6.1}$$

where $\boldsymbol{\theta}_q$ denote the trainable parameters of the network. The bidirectional RNN consists of two RNNs: the forward RNN that reads the question starting from the first word until the last word and encodes it as a sequence of forward context vectors $(\overrightarrow{\boldsymbol{h}_1^q}, \overrightarrow{\boldsymbol{h}_2^q}, ..., \overrightarrow{\boldsymbol{h}_k^q})$, and the reverse RNN that encodes the question starting from the last word until the first word: $(\overleftarrow{\boldsymbol{h}_k^q}, \overleftarrow{\boldsymbol{h}_{k-1}^q}, ..., \overleftarrow{\boldsymbol{h}_1^q})$. The resulting context vectors are concatenations of the forward and reverse context vectors at each step, i.e.

[1] We initially experimented with a unidirectional RNN too, and the bidirectional was clearly superior, so we only report the results when the bidirectional RNN is used.

$\boldsymbol{h}_i^q = [\overrightarrow{\boldsymbol{h}_i^q}, \overleftarrow{\boldsymbol{h}_i^q}]$.

As the encoded vector representation of the question, we use the concatenation of the last context vector of the forward RNN, i.e. corresponding to the last word, and the last context vector of the backward RNN, i.e. corresponding to the first word, as is usually done in the encoder-decoder architecture (Bahdanau et al., 2014):

$$\boldsymbol{enc}^q = [\overrightarrow{\boldsymbol{h}_k^q}, \overleftarrow{\boldsymbol{h}_1^q}] \tag{6.2}$$

The second bidirectional RNN encodes the answer in the same way:

$$f_{RNN}^a(\boldsymbol{w}_i^a, \boldsymbol{\theta}_a) = \boldsymbol{h}_i^a \tag{6.3}$$

$$\boldsymbol{enc}^a = [\overrightarrow{\boldsymbol{h}_p^a}, \overleftarrow{\boldsymbol{h}_1^a}] \tag{6.4}$$

where $\boldsymbol{\theta}_a$ denote the trainable parameters of the network. Figure 6.1 illustrates the RNN-MLP system.

6.1.1 Prediction and Training

The score for the given question-answer pair is predicted with an MLP:

$$y = f_{MLP}([\boldsymbol{enc}^q, \boldsymbol{enc}^a], \boldsymbol{\theta}_s) \tag{6.5}$$

where $\boldsymbol{\theta_s}$ denote the trainable parameters of the network.

The network is trained by minimizing cross-entropy:

$$L(y, \boldsymbol{\theta}) = -\bar{y}\log(y) - (1-\bar{y})\log(1-y)$$

where $\boldsymbol{\theta}$ are all network's parameters, i.e. $\boldsymbol{\theta}_q, \boldsymbol{\theta}_a, \boldsymbol{\theta}_s$ and $\bar{y}$ is the true label (0 or 1):

$$\bar{y} = \begin{cases} 1 & \text{if } a \text{ is the best answer of the question } q \\ 0 & \text{otherwise} \end{cases}$$

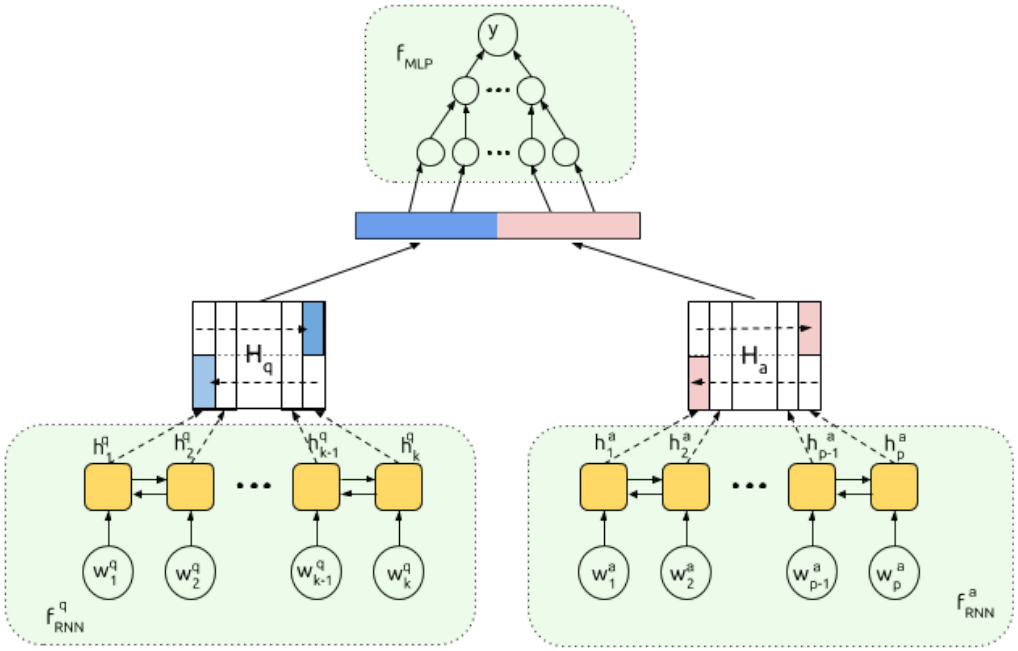

Figure 6.1: RNN-MLP model for answer ranking. Given a question-answer pair, two separate RNNs are used to encode the question and the answer, and the encodings are concatenated and passed to an MLP. All three networks are trained together.

6.1.1.1 Hyperparameters

For this set of experiments we set the dimensionality of word embeddings and the dimensionality of RNN states to 100. The MLP has five hidden layers. We set the number of units in each hidden layer depending on the number of inputs to that layer: for a hidden layer $i = 0, 1, 2, 3, 4$ with m inputs we set the number of units to:

$$\text{num\_units} = \min(m/2, 2^{12-i}) \tag{6.6}$$

This means that if an RNN has a 100-dimensional state, the question encoding and the answer encodings have 200 dimensions, as we use a bidirectional RNN. Then, the input to the MLP for the RNN-MLP-last system has 400 dimensions, and the hidden layers have 200, 100, 50 and 25 units.

The lengths of questions and answers in our dataset vary. To handle this, we

use the dynamic bidirectional RNN[2], which can handle sequences of variable length. However, we still need to pad the questions and answers. As we use mini-batch training, we pass mini-batches as three-dimensional arrays, thus, within a mini-batch all sequences should be of the same length. We also specify the lengths of all instances, allowing the RNN to ignore whatever is further than the length of that instance, i.e. padding. We train the model using SGD. The mini-batch size is set to 100. The L2 regularisation rate and dropout probability after initial tuning were set to 10^{-7} and 0.2 for the YA datasets and to 10^{-6} and 0.3 for the Ask Ubuntu dataset, as the number of parameters is higher on the latter (see Table 6.1). We evaluate the model on the development set every 500 iterations, and stop the training if the development loss does not decrease for 10 consecutive evaluations.

RNN cell	**Yahoo**	**Ask Ubuntu**
GRU	1.83M	4.10M
LSTM	1.91M	4.18M

Table 6.1: Number of trainable parameters of the RNN-MLP model with the maximal question lengths set to 15 and 150 for the YA and the AU datasets, the answer lengths are set to 100. The word embeddings and the RNN state dimensionalities are set to 100. The size of the MLP is calculated as in Equation 6.6.

6.1.2 LSTM versus GRU for Answer Ranking

In Chapter 2 we described two variants of recurrent neural networks that use the gating mechanism: long short term memory (LSTM) networks and gated recurrent networks (GRU for Gated Recurrent Units). The gating mechanism enables the ability to represent long-term dependencies, hence the popularity of LSTMs and GRUs in language processing where capturing long-term dependencies is of high importance. The essential difference between the two cells is that the LSTM has three gates and a separate memory state and a hidden state, while the GRU has only two gates and also merges the two states into one, and thus, has fewer trainable parameters. In Table 6.1 we report the number of trainable parameters of the full

[2]implemented as a part of the tensorflow library: `http://www.tensorflow.org`

Yahoo! Answers		
Encoder	**P@1**	**MRR**
LSTM	**37.45**†	**58.12**
GRU	36.73	57.06
Paragraph Vector	37.37	57.05
Random baseline	15.74	37.40
CR baseline	22.63	47.17
Jansen et al. (2014)	30.49	51.89
Fried et al. (2015)	33.01	53.96
Ask Ubuntu		
Encoder	**P@1**	**MRR**
LSTM	**42.64**†	**65.28**
GRU	41.96	64.87
Paragraph Vector	41.48	64.33
Random baseline	26.60	53.64
CR baseline	35.36	60.17
Chronological baseline	37.68	60.06

Table 6.2: Performance of the GRU and the LSTM encoders versus the baselines for answer ranking. The improvement of the LSTM model over the GRU model is not statistically significant ($p > 0.05$).

network including the two encoders and the MLP predictor. Several studies have compared the two variants on various tasks, and most have found they perform similarly well and outperform the vanilla *tanh* RNN (Chung et al., 2014; Jozefowicz et al., 2015).

Table 6.2 compares the performances of the GRU and the LSTM encoders against the baselines on the task of answer ranking. On both datasets, the two encoders outperform most baselines, with the only exception being the Paragraph Vector model on the YA dataset, where it performs at the level of the LSTM encoder and outperforms the GRU encoder. This is an interesting observation, that the PV model trained in an unsupervised fashion, makes a very strong baseline able to compete with much more sophisticated encoders. This is probably due to its ability to leverage the large amounts of unlabelled data to obtain powerful representations of the documents.

Comparing the performance of the LSTM and the GRU encoders, we can see from the experimental results, that they provide similar results, with the LSTM providing an insignificant improvement over the GRU model.

6.1.3 Augmenting the Representations

In the set of experiments described above we used two separated encoders for the question and the answer. We do not directly pass any information about the question, while encoding the answer and vice versa and let the final deep MLP decide how relevant the encoded answer is to the encoded question, i.e. the interactions between the question and the answer are implicit, as we train the three networks (the encoders and the MLP) together. In order to make the interactions between the question and the answer more direct, we adopt the attention mechanism of Bahdanau et al. (2014). This mechanism was introduced for the encoder-decoder architecture for the task of machine translation. We use this mechanism in the following way: (1) the question is encoded using an LSTM as in Equation 6.1:

$$f^q_{RNN}(\boldsymbol{w}^q_i, \boldsymbol{\theta}_q) = \boldsymbol{h}^q_i \tag{6.7}$$

(2) while encoding the answer, we use the attention over the context vectors of the question encoder, as described in Chapter 2 (Section 2.3.3.4):

$$f^a_{RNN}(\boldsymbol{w}^a_i, \boldsymbol{\theta}_a, \boldsymbol{h}^q_i) = \boldsymbol{h}^a_i \tag{6.8}$$

Similar attention mechanisms were used by dos Santos et al. (2016). The intuition behind using attention in this architecture is that certain parts of the answer may be aligned to certain parts of the question, which happens often in factoid answer selection, see an example from the TREC QA dataset in Figure 6.2.

We also explore a much simpler way to explicitly encode the interaction between the question and the answer. Instead of using the output of the question encoder

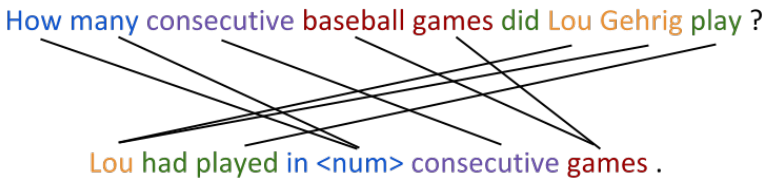

Figure 6.2: Example of a factoid question from TREC QA dataset with its correct answer and a possible alignment between them. Note that attention mechanisms define a *soft* alignment rather than the precise alignment represented here.

when encoding the answer, we encode them separately and then apply the *interaction transformation* to the context vectors. More specifically, let $\boldsymbol{H}_q$ denote the matrix composed of the outputs of the question encoder RNN:

$$\boldsymbol{H}_q = \begin{pmatrix} h^q_{1,1} & h^q_{1,2} & \cdots & h^q_{1,k} \\ h^q_{2,1} & h^q_{2,2} & \cdots & h^q_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ h^q_{d,1} & h^q_{d,2} & \cdots & h^q_{d,k} \end{pmatrix}$$

and $\boldsymbol{H}_a$ denote the matrix composed of the outputs of the answer RNN:

$$\boldsymbol{H}_a = \begin{pmatrix} h^a_{1,1} & h^a_{1,2} & \cdots & h^a_{1,p} \\ h^a_{2,1} & h^a_{2,2} & \cdots & h^a_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ h^a_{d,1} & h^a_{d,2} & \cdots & h^a_{d,p} \end{pmatrix}$$

d is a dimensionality parameter to be experimentally tuned. We calculate the similarity matrix $\boldsymbol{S}$ between $\boldsymbol{H_q}$ and $\boldsymbol{H_a}$, so that each element s_{ij} of the $\boldsymbol{S}$ matrix is a dot product between the corresponding encodings:

$$s_{ij} = \boldsymbol{h}^q_i \cdot \boldsymbol{h}^a_j$$

The similarity matrix $\boldsymbol{S}$ is unrolled and passed to the multilayer perceptron along

with the question and answer encodings:

$$y = f_{MLP}([\boldsymbol{enc}^q, \boldsymbol{enc}^a, \boldsymbol{S}], \boldsymbol{\theta}_s) \tag{6.9}$$

This approach requires us to pad all questions and answers to the same length. This is illustrated in Figure 6.3.

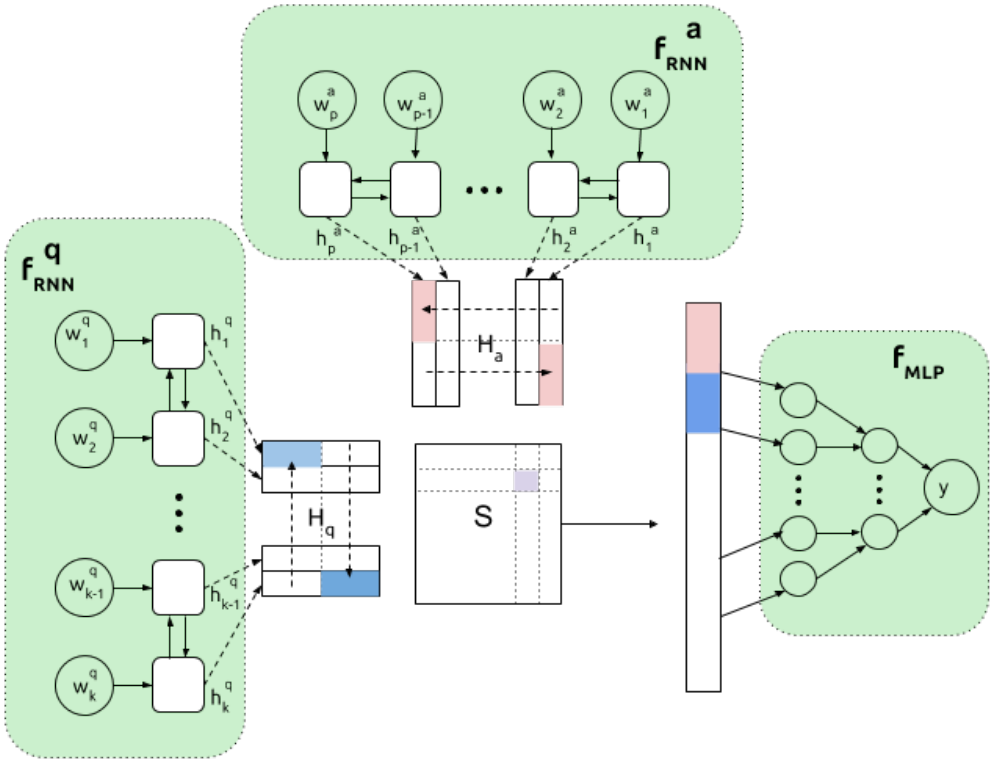

Figure 6.3: RNN-MLP model for answer ranking that uses the interaction features. Given a question-answer pair, two separate RNNs are used to encode the question and the answer, and the encodings are concatenated and passed to an MLP, as well as the similarity matrix calculated as the pairwise dot product of the question and the answer encodings. All three networks are trained together.

In Table 6.3 we compare the performance of a standard LSTM versus the LSTM that uses the attention and the LSTM that uses the interaction transformation. The attention mechanism we borrowed from the encoder-decoder architecture does not improve the performance of a plain LSTM encoder. Unlike the neural machine translation settings, there is no actual alignment between the question and the answer, and a more suitable attention mechanism may be needed for this task. The

interaction transformation also does not improve the performance of the LSTM. A possible reason for that is the varying lengths of the questions and the answers, that require the questions and the answers to be padded in order to apply the transformation, thus, a lot of zeros are passed to the MLP.

Yahoo! Answers		
Encoding	**P@1**	**MRR**
standard	**37.45**	**58.12**
S-matrix	36.42	56.62
attention	37.22	58.04
Ask Ubuntu		
Encoding	**P@1**	**MRR**
standard	**42.64**	**65.28**
S-matrix	40.36	64.22
attention	40.64	63.91

Table 6.3: Comparison of variations of encodings with LSTMs for answer ranking. *Standard* encoding uses the last outputs of the forward and the backward LSTMs; *S-matrix* adds the interaction transformation features to the standard encoding; *attention* adapts the attention mechanism of Bahdanau et al. (2014)

6.2 Answer Ranking with Convolutional Neural Networks

In the previous section we reported answer ranking experiments, where we used an RNN to encode questions and answer, i.e. represent them as fixed-sized vectors. Besides an RNN, a convolution neural network (CNN) can also be used to encode a text. Initially designed for computer vision tasks, the CNNs became very popular in the area of NLP and were applied to answer selection (Severyn and Moschitti, 2015a; Tymoshenko et al., 2016a; dos Santos et al., 2016), sentiment analysis (Kim, 2014; dos Santos and Gatti, 2014; Kalchbrenner et al., 2014) and question type classification (Kim, 2014; Kalchbrenner et al., 2014). We have already used a convolutional architecture in Chapter 3 for the task of semantically equivalent question detection. In this chapter, we use an extended version of this architecture, i.e. we use various

filter sizes, and use a better regularisation. We use a convolution architecture similar to the one presented by Kim (2014). Let x be a text, e.g. a question or an answer, and $\boldsymbol{x}_i \in \mathbb{R}^k$ the embedding of the i-th word in the text, i.e.

$$x = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$$

A **filter** of size h is a vector $\boldsymbol{w} \in \mathbb{R}^{hk}$ which is applied to a word window of size h and produces a **feature** c_i:

$$c_i = f(\boldsymbol{w}[\boldsymbol{x}_i, ..., \boldsymbol{x}_{i+h-1}] + b) \tag{6.10}$$

where $b \in \mathbb{R}$ is a bias and f is a non-linear function, such as the ReLU or the hyperbolic tangent. The filter is applied to every possible word window of size h, and the vector of the produced features is called a **feature map**:

$$\boldsymbol{c} = (c_1, c_2, ..., c_{n-h+1}) \tag{6.11}$$

After that, a max-pooling operation takes the maximum from each feature map:

$$\hat{c} = max(c_1, c_2, ..., c_{n-h+1}) \tag{6.12}$$

The intuition behind the max-pooling operation is to capture the most important information from each feature map (Kim, 2014). Usually, not just one but a number of filters m is applied to each window, i.e.:

$$\boldsymbol{c}_i = f(\boldsymbol{W}^\top[\boldsymbol{x}_i, ..., \boldsymbol{x}_{i+h-1}] + \boldsymbol{b}) \tag{6.13}$$

where $\mathbf{W}$ is a matrix of size $hk \times m$ and $\boldsymbol{b} \in \mathbb{R}^m$ is a bias vector. The representation of the text x is obtained with max-pooling:

$$\boldsymbol{e} = (max(\boldsymbol{c}_1), ..., max(\boldsymbol{c}_m)) \tag{6.14}$$

Figure 6.4 illustrates the CNN that we use to encode a question or an answer. The main difference from the architecture used in Chapter 3 is the use of various filters, i.e. word window, sizes instead of using only one filter of a fixed size. We also train the system differently. We use two separate CNNs to encode the question and the answer, then the representations are concatenated and passed to an MLP. The network is trained in the same way as the RNN-based system described in Section 6.1, i.e. by minimising cross-entropy on the training set.

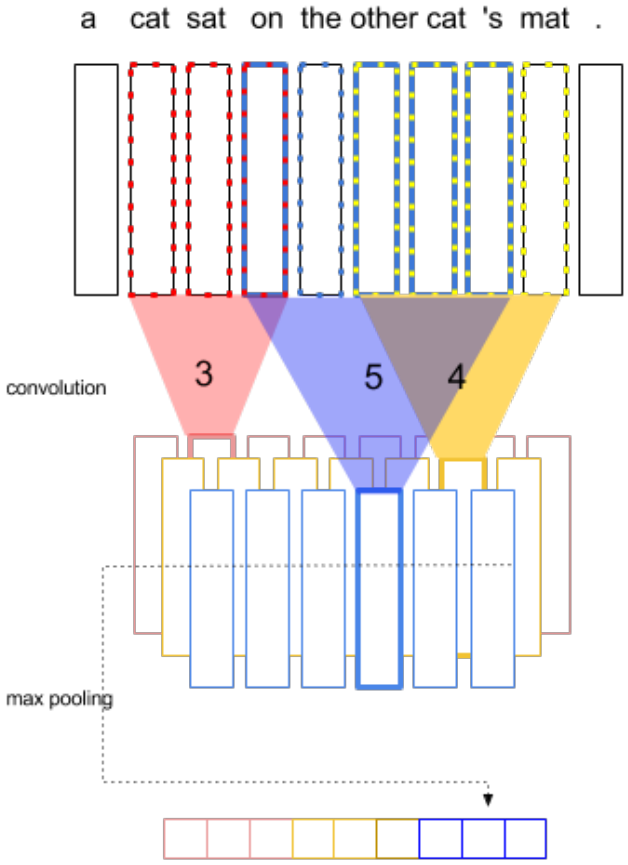

Figure 6.4: Illustration of a CNN encoder for answer ranking. First, words are represented as word embeddings. Second, a convolution with multiple filter sizes is applied to the word embeddings. Finally, max-pooling is applied to the output of the convolutions.

6.2.1 Hyperparameters

We experiment with filter (word window) sizes from one to five, and set the number of filters of each size to 100. The dropout probability was set to 0.2 for the CNN and the MLP, and the L2 regularisation rate was set to 10^{-7} on the YA dataset, and to 0.3 and 10^{-6} on the AU dataset. The model was trained with SGD with a mini-batch of size 100 and evaluated on the development set every 500 steps. The training was stopped if there was no improvement on the development set for 10 consecutive evaluations.

Yahoo! Answers		
Encoder	**Test P@1**	**Test MRR**
LSTM	**37.45***	**58.12**
CNN	35.45	55.98
Paragraph Vector	37.37	57.05
Random baseline	15.74	37.40
CR baseline	22.63	47.17
Jansen et al. (2014)	30.49	51.89
Fried et al. (2015)	33.01	53.96
Ask Ubuntu		
Encoder	**Test P@1**	**Test MRR**
LSTM	**42.64***	**65.28**
CNN	34.76	59.96
Paragraph Vector	41.48	64.33
Random baseline	26.60	53.64
CR baseline	35.36	60.17
Chronological baseline	37.68	60.06

Table 6.4: Answer ranking performance when using the RNN versus the CNN encoder. *The improvement is statistically significant ($p < 0.05$).

6.2.2 CNN versus RNN for Answer Ranking

We apply the CNN with the hyperparameters providing the highest development P@1 to the test set. Table 6.4 reports the performance of the CNN versus the best performing RNN-based system described in the previous section. On the YA dataset, the CNN proves competitive with the RNN, although, the LSTM produces significantly better results. However, on the AU dataset, the CNN performs simi-

larly to the candidate retrieval baseline and is far below the RNN-based systems. The explanation for this is that the AU dataset contains much longer questions and answers. The CNNs are in some sense similar to the n-gram model: they encode local features. Unlike the RNNs, they lack the ability to represent long-term dependencies that is essential when encoding long texts. Nonetheless, the CNNs succeed in encoding sentences (dos Santos and Gatti, 2014; Kim, 2014) or short texts, e.g. Twitter data (Severyn and Moschitti, 2015b; Kalchbrenner et al., 2014). A two-level CNN like the one presented by Denil et al. (2014) that first composes words into sentences and then, sentences into documents, might be a better variation of a CNN architecture for longer texts.

6.3 Multi-Channel Recurrent Convolutional Neural Network

In the previous sections of this chapter, we have explored RNNs and CNNs as the encoders in the task of answer ranking. Previous studies on non-factoid answer ranking found that discourse information, i.e. the information on the boundaries of possible discourse segments, helps to match the question and the answer (Jansen et al., 2014; Verberne et al., 2007). While the RNNs provide powerful text representations their recurrent nature does not allow them to detect clause boundaries. On the other hand, the CNNs may be more suitable to encode discourse segments, as the convolution captures every possible word window of determined sizes. However, the CNNs fail to represent long-term dependencies. In this section we propose a novel architecture called **Multi-Channel Recurrent Convolutional Neural Network (MC-RCNN)** that is aimed to overcome these drawbacks of the two models by combining them. It first uses a forward and a backward RNN to encode the sequence, and then applies a multi-channel CNN treating each direction's output as a channel. In other words, the first channel on the CNN consists of the outputs of the forward RNN encoder, its second channel consists of the outputs of

the backward RNN encoder, the third channel receives word embeddings like in a typical CNN encoder setup. The last states of each of the two RNNs and the output of the CNN are then passed to a multilayer perceptron. Figure 6.5 illustrates this architecture.

More formally, the word embeddings $(\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_n)$ are encoded into a sequence of forward output vectors using a forward RNN:

$$f_{forwRNN}(\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_n, \boldsymbol{\theta}_{forw}) = (\overrightarrow{\boldsymbol{h}_1}, \overrightarrow{\boldsymbol{h}_2}, ..., \overrightarrow{\boldsymbol{h}_n}) \tag{6.15}$$

and into a sequence of backward output vectors using a backward RNN:

$$f_{backRNN}(\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_n, \boldsymbol{\theta}_{back}) = (\overleftarrow{\boldsymbol{h}_n}, \overleftarrow{\boldsymbol{h}}_{n-1}, ..., \overleftarrow{\boldsymbol{h}_1}) \tag{6.16}$$

where $\boldsymbol{\theta}_{forw}$ and $\boldsymbol{\theta}_{back}$ are trainable parameters of the networks.

Then the word embeddings, the forward output vectors and the backward output vectors represent the three channels of the CNN input layer, and the encoding is produced is follows:

$$\begin{aligned} f_{CNN}([&\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_n], \\ &[\overrightarrow{\boldsymbol{h}_1}, \overrightarrow{\boldsymbol{h}_2}, ..., \overrightarrow{\boldsymbol{h}_n}], \\ &[\overleftarrow{\boldsymbol{h}_n}, \overleftarrow{\boldsymbol{h}}_{n-1}, ..., \overleftarrow{\boldsymbol{h}_1}], \boldsymbol{\theta}_{CNN}) = \boldsymbol{enc}_{CNN} \end{aligned} \tag{6.17}$$

The final MLP predictor then receives both the encoding produced by the CNN and the encodings produced by the RNNs, i.e.

$$f_{MLP}(\boldsymbol{enc}_{CNN}, \overrightarrow{\boldsymbol{h}_n}\overleftarrow{\boldsymbol{h}_1}, \boldsymbol{\theta}_{MLP}) = \boldsymbol{y} \tag{6.18}$$

We train this model in the same way as described in Section 6.1. As we can see from Table 6.5, the MC-RCNN performs at the level of the RNN models on the YA dataset, bringing only an insignificant improvement when a GRU cell is used. On the contrary, on the AU dataset, the MC-RCNN architecture with either the LSTM

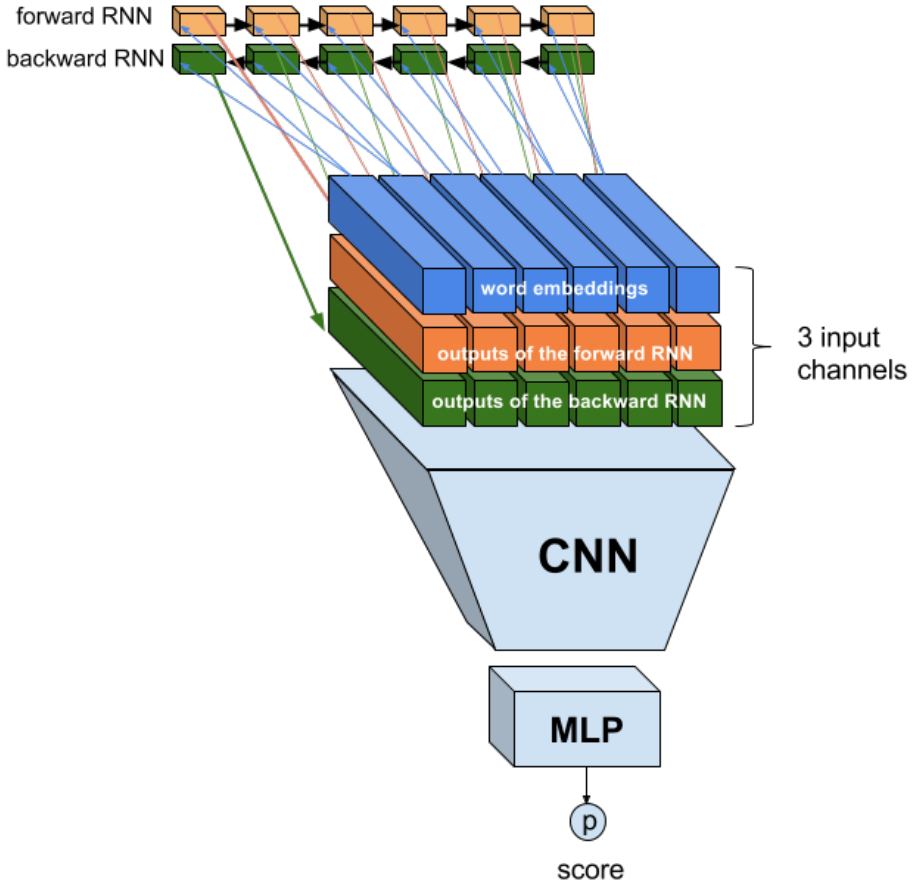

Figure 6.5: Illustration of MC-RCNN model. First, the input sequence is encoded with a forward RNN and a backward RNN. The outputs of the two RNNs and the original word embeddings are the three inputs channels of a CNN. The CNN outputs is then passed to an MLP that predicts a score.

or the GRU cell outperforms the RNN and the CNN encoders. This shows that this model is particularly suitable for longer texts.

6.4 Summary

In this chapter we explored recurrent and convolutional neural networks for the purposes of encoding the question and the answer for answer ranking. We have also proposed a novel architecture called multi-channel recurrent convolutional neural network, that applied a three-channel CNN to the original word embedding representation as well as the outputs of the forward and the backward RNNs. Our experiments show that:

Yahoo! Answers			
Model	**Cell**	**P@1**	**MRR**
MC-RCNN	LSTM	36.25	56.66
MC-RCNN	GRU	**37.62**†	57.60
RNN	LSTM	37.45	**58.12**
RNN	GRU	36.73	57.06
CNN	-	35.45	55.98
Ask Ubuntu			
Model	**Cell**	**P@1**	**MRR**
MC-RCNN	LSTM	43.56	66.14
MC-RCNN	GRU	**44.36***	**66.39**
RNN	LSTM	42.64	65.28
RNN	GRU	41.96	64.87
CNN	-	34.76	59.96

Table 6.5: Performance of the system with a MC-RCNN encoder versus the RNN and the CNN-based systems. *The improvement over the RNN and CNN based systems on the AU dataset is statistically significant ($p < 0.05$). †The improvement over the RNN and CNN based systems on the YA dataset is not statistically significant ($p > 0.05$).

- LSTMs generally perform slightly better than GRUs when combined with an MLP for the task of answer ranking however the improvements are not statistically significant;
- Incorporating the attention mechanism of Bahdanau et al. (2014) which is successful in neural machine translation, into the LSTM-based answer ranking does not improve the performance. This is probably due to the fact that the question and the answer are not aligned unlike in machine translation.
- CNNs perform comparably to RNNs when the questions and answers are relatively short. However, they are not suitable for encoding long questions and answers.
- MC-RCNN outperforms the CNNs and in most cases the LSTMs too. The improvement is more remarkable on the Ask Ubuntu dataset which has more complex questions and answers.
- MC-RCNN performs better in combination with the GRU, unlike the RNN that achieves better results when using the LSTM cell. This needs further investigation.

Chapter 7

Further Analysis of Answer Ranking

In previous chapters we explored different ways to represent questions and answers for the task of answer reranking. Firstly, in Chapter 5 we evaluated the Paragraph Vector model (Le and Mikolov, 2014). This approach is simple and achieves good results, but requires a big in-domain corpus for pretraining and either assumes that the test data is available at pretraining time, or requires inference with a gradient-based method at test time. Next, in Chapter 6 we learned the representations for questions and answers with RNNs, CNNs and MC-RCNNs as part of the task.

In this chapter, we rather briefly explore various aspects of answer ranking that we did not discuss in previous chapters. In particular, we focus on the following questions: (1) can the performance of a neural system can be improved by inclusion of tried-and-tested features? (2) can character-level representations help to overcome the specificity of noisy user-generated content? (3) can unsupervised pretraining improve the performance of the proposed neural systems? (4) do the models trained and tuned on the YA and AU datasets perform well on a different dataset?

This chapter is structured as follows: We experiment with character-level embeddings instead of word-level embeddings in Section 7.1. In Section 7.2 we combine the RNN and the CNN-based approaches with the discourse features that were in-

troduced by Jansen et al. (2014). In Section 7.3 we empirically investigate the impact of pretrained word embeddings on the performance of our models. In Section 7.4 we provide an error analysis of the best performing systems. In Section 7.5 we evaluate our neural models on the dataset of the SemEval 2016 shared task on answer ranking. Finally, we discuss some open questions and draw conclusions in Section 7.6.

7.1 Character-level versus Word-level Embeddings

Many state-of-the-art models in NLP rely on character-level embeddings instead of (Kim et al., 2016) or as well as (dos Santos and Zadrozny, 2014) word embeddings. The main intuition behind using character-level embeddings is that they can capture morphological information and better deal with out-of-vocabulary and misspelled words. The use of character embeddings instead of word embeddings also obviates the need to tokenise the data. At the same time, it slows down the RNN training significantly, as the sequences become much longer, i.e. the number of RNN steps increases. Consider the following example: *I shot an elephant in my pyjamas*. A word-level RNN would need seven steps to encode it, whereas a character-level RNN would need 32 steps. Another drawback of the character-level representations is that they do not have explicit knowledge of word boundaries, and this makes them usually less effective than similar word-level models (Sutskever et al., 2011).

We set the dimensionality of character embeddings to 20, all other hyperparameters remain the same as when training a word-level model. We compare the performance of several of the presented models with character-level instead of word-level embeddings in Table 7.1. The performance of all RNN-based models drops significantly when using character-level representations. However, when a CNN is used to encode questions and answers, its performance drops less notably on the YA data and even increases on the AU dataset. Moreover, on character-level representations the CNN proves superior to RNN. However, its performance is still below

Yahoo			
Encoder	**Representation**	**P@1**	**MRR**
CNN	word	**35.45***	**55.98**
CNN	char	32.01	53.53
GRU	word	**35.77***	**56.97**
GRU	char	30.21	52.15
LSTM	word	**37.45***	**58.12**
LSTM	char	30.57	52.52
Ask Ubuntu			
Encoder	**Representation**	**P@1**	**MRR**
CNN	word	34.76	56.96
CNN	char	**38.40***	**62.33**
GRU	word	**41.96***	**64.87**
GRU	char	37.12	61.08
LSTM	word	**42.64***	**65.28**
LSTM	char	37.04	61.10

Table 7.1: Answer reranking performance of different models when using word-level versus character-level embeddings. For the GRU and LSTM we use the *last* encoding for faster training. *All improvements are statistically significant.

the ones of the RNN-based word-level systems.

7.2 Injecting Discourse Features into the Neural System

We propose to enrich the neural model presented in Section 6.1 with additional features. In the systems described in the previous chapter, the encoded vectors produced by an RNN, a CNN or an MC-RCNN are concatenated and passed to an MLP. We suggest concatenating these encodings with additional external features, e.g. discourse features, and then passing them to the MLP. More specifically, the score is now predicted by the following function (compare to Equation 6.5).

$$y = f_{MLP}([\boldsymbol{enc}^q, \boldsymbol{enc}^a, \boldsymbol{x}_{ext}], \boldsymbol{\theta}_s) \tag{7.1}$$

where $\boldsymbol{x}_{ext}$ is the vector of additional features. Figure 7.1 illustrates the model with the additional discourse features. An RNN, a CNN or a MC-RCNN can be

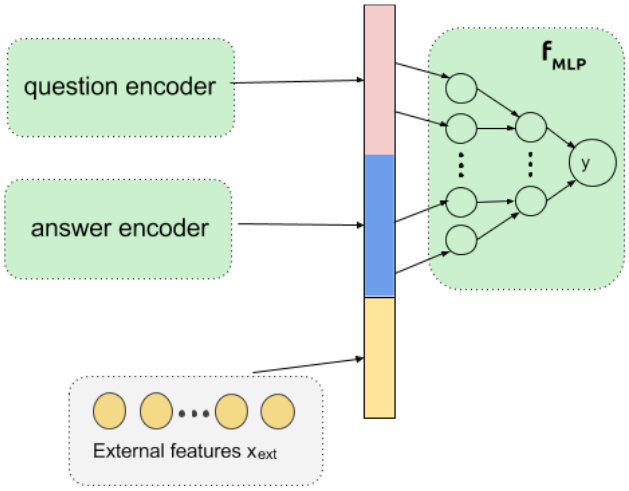

Figure 7.1: Illustration of the architecture that incorporates additional features. The question encoder can be, for instance, an RNN, a CNN or a MC-RCNN. As the additional features, in our experiments we use the features produced by the discourse marker model presented in Jansen et al. (2014), however, other features can also be incorporated.

used as a question and an answer encoder. In the following sections we present experiments where the model is enriched with discourse features produced by the discourse marker model of Jansen et al. (2014).

7.2.1 Discourse Features

Based on the intuition that modelling question-answer structure goes beyond sentence level, Jansen et al. (2014) propose an answer ranking model based on discourse markers combined with lexical semantic information. We inject the features produced by their discourse marker model (DMM) combined with their lexical semantics model (LS) into the neural system we described in previous chapters. The DMM model is based on the findings of Marcu (1997), who showed that certain cue phrases indicate boundaries between elementary textual units with sufficient accuracy. These cue phrases are further referred to as discourse markers. For English, these markers include *by, as, because, but, and, for* and *of* – the full list can be

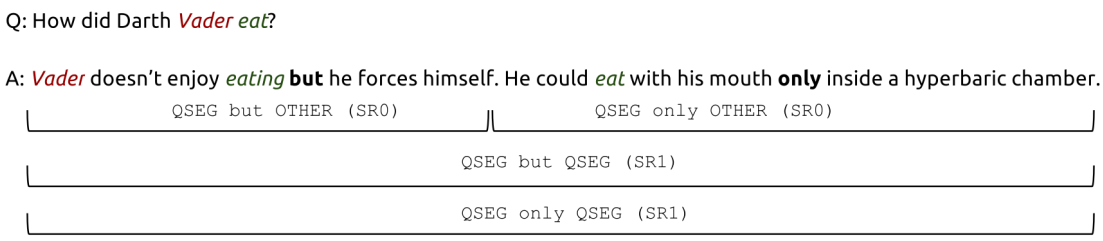

Figure 7.2: Feature generation for the discourse marker model of Jansen et al. (2014): first, the answer is searched for the discourse markers (in **bold**). For each discourse marker, there are several features that represent whether there is an overlap (`QSEG`) with the question before and after the discourse marker. The features are extracted for sentence range from 0 (the same range) to 2 (two sentences before and after).

found in Appendix B in Marcu (1997).

We illustrate the feature extraction process of Jansen et al. (2014) in Figure 7.2. First, the answer is searched for discourse markers. Each marker divides the text into two arguments: preceding and following the marker. Both arguments are searched for words overlapping with the question. Each feature denotes the discourse marker and whether there is an overlap with the question (`QSEG`) or not (`OTHER`) in the two arguments defined by the marker. The sentence range (`SR`) denotes the length (in sentences) of the marker's arguments. For example, `QSEG by OTHER SR0` means that in the sentence containing the *by* marker there is an overlap with the question before the marker and there is no overlap with the question after the marker. This results in 1384 different features. To assign values to each feature, the similarity between the question and each of the two arguments is computed, and the average similarity is assigned as the value of the feature. Jansen et al. (2014) use cosine similarity over *tf.idf* and over the vector space built with a skip-gram model (Mikolov et al., 2013b).

7.2.1.1 Results: discourse features

In Table 7.2 we report the results for the systems enhanced with the discourse features and the discourse features on their own with an MLP (MLP-Discourse). The MLP-Discourse outperforms the random and the CR baselines for both datasets. It

also perform better than the approach of Jansen et al. (2014) who used SVMrank with a linear kernel. This might be due to the ability of the MLP to model non-linear dependencies. However, this model's performance is below the RNN, the CNN and the MC-RCNN performances on their own without any external features.

The inclusion of the discourse features improves the performance of the LSTM, the GRU and the CNN encoders on both datasets. However, on the YA dataset, the improvements are not statistically significant. On the AU dataset, the improvements are statistically significant with $p < 0.05$. The improvement is especially notable when the CNN encoder is used on the AU dataset. The CNN encoder performed poorly on its own on this dataset, but its performance was drastically improved by inclusion of the discourse features. This is possibly because the discourse information helps the CNN to overcome its inability to account for long-term dependencies.

The performance of the MC-RCNN is improved by the inclusion of the discourse features only when the LSTM cell is used, and the improvement is not statistically significant. The MC-RCNN does not seem to benefit from the discourse information, suggesting that the discourse features do not provide any extra information that is not captured already by the model. However, this may also mean that some sort of feature normalisation is required, e.g. normalising the discourse vector and the output of the encoder separately and then, perhaps, normalising the concatenation.

Manual error analysis shows that the improvement brought by the discourse features to most models is due to a better handling of the questions with long answers. In certain cases, where the best answer is relatively long, the RNN model assigned a higher score to a shorter answer.

7.3 The Impact of Pretrained Word Embeddings

All the RNN and CNN-based models for answer reranking we experimented with in Chapter 6 did not use any external corpora for pretraining the word embeddings, i.e. the word embeddings were initialised by sampling from the random uniform

Yahoo			
Encoder	**Discourse**	**P@1**	**MRR**
CNN	No	35.45	55.98
CNN	Yes	**35.73**†	**55.91**
LSTM	No	37.45	58.12
LSTM	Yes	**38.02**†	**58.26**
MC-RCNN	No	**37.62**†	**57.60**
MC-RCNN	Yes	37.21	57.50
MLP-Discourse		32.72	53.54
Ask Ubuntu			
Encoder	**Discourse**	**P@1**	**MRR**
CNN	No	34.76	59.96
CNN	Yes	**41.72***	**64.59**
LSTM	No	42.64	65.28
LSTM	Yes	**43.80***	**66.20**
MC-RCNN	No	**44.36**†	**66.39**
MC-RCNN	Yes	43.36	65.80
MLP-Discourse		37.80	61.75

Table 7.2: Experimental results on the test set for different encoders with and without discourse features.

Corpus	**Number of tokens**	**Vocabulary size**
Google News	100B	692K
L6 Yahoo! Answers	1.9B	1.2M
Ask Ubuntu dump	97M	183K

Table 7.3: Details on the corpora used to pretrain the skip-gram model.

distribution around zero. It has been shown that unsupervised pretraining helps deep learning due to its ability to provide better initialisation and also serve as a regulariser and prevent overfitting (Erhan et al., 2010). In Section 5 we also showed that distributed representations for documents obtained in an unsupervised manner using the Paragraph Vector model were useful for the task of answer reranking. In this section we compare the best performing neural models on the YA and AU datasets when the word vectors are (1) initialised randomly and (2) pretrained using the skip-gram model (Mikolov et al., 2013a) on (2a) in-domain data and (2b) out-of-domain data. To train domain-specific embeddings we use the L6 dataset of Yahoo! Answers questions and the Ask Ubuntu September 2014 data dump. The YA and the AU datasets were originally sampled from these two corpora respectively.

To evaluate the impact of pretraining on out-of-domain data, we choose to use publicly available word vectors trained on about 100 billion words from Google News created by Mikolov et al. (2013b). Some statistics on the corpora are presented in Table 7.3. Before training the models, we tokenise the data with the tokeniser packaged with Stanford parser and lowercase the tokenised data. We use the original word2vec implementation[1] of the skip-gram model. We subsample frequent words, as suggested by Mikolov et al. (2013b): every word is sampled with the following probability:

$$P(w) = 1 - \sqrt{\frac{t}{f(w)}} \tag{7.2}$$

where $f(w)$ is the frequency of the word w and t is the subsampling rate. We set the subsampling rate to 10^{-4}. Words with fewer than five occurrences were ignored. The models were trained for 15 iterations using negative sampling with 25 negative examples. The word window was set to 5.

We select the best performing models on the YA and the AU datasets, i.e. the LSTM-MLP with and without the additional discourse features on the YA dataset and the MC-RCNN model with the GRU cell for the AU dataset and compare their performances with the word embeddings randomly initialised versus the ones pretrained with the skip-gram model, as described above.

Table 7.4 shows the performance of these models. Surprisingly, the unsupervised pretraining did not improve the performance of the best model (LSTM-MLP-discourse for the YA dataset and MC-RCNN model for the AU dataset) on either of the datasets. However, pretraining on domain-specific data does improve the performance of the LSTM-MLP model when no extra features are used. On the AU dataset, pretraining on in-domain data does not improve the performance but neither does it worsen it significantly, as it remains almost the same as with randomly initialised embeddings.

When pretraining on out-of-domain data (Google News) the performance on

[1] `https://code.google.com/archive/p/word2vec/`

Yahoo: LSTM-MLP		
Embeddings	**P@1**	**MRR**
Random	37.45	58.12
Yahoo	**38.70**	**59.14**
Google	36.69	57.25
Yahoo: LSTM-MLP-discourse		
Embeddings	**P@1**	**MRR**
Random	**38.02**	**58.26**
Yahoo	**38.02**	58.24
Google	37.15	57.10
Ask Ubuntu: MC-RCNN (GRU)		
Embeddings	**P@1**	**MRR**
Random	**44.36**	**66.39**
Ask Ubuntu	44.04	66.22
Google	38.80	61.96

Table 7.4: Performance of the best performing models with random and pretrained embeddings.

the YA dataset slightly decreases, however, on the AU dataset it drops drastically. This is probably due to significant differences between the AU dataset, which is a technical CQA and the pretraining corpus sampled from Google News. Apparently, the YA data differs from the newswire less dramatically and the differences might be slightly mitigated by the large amounts of data in the Google News corpus.

Overall, our results suggest that unsupervised pretraining does not help the hybrid model and the LSTM-MLP model that uses the discourse information. It does improve the performance of the LSTM-MLP model with no extra features but only when in-domain data is used for pretraining. This suggests that the extra discourse features do not allow us to benefit from unsupervised pretraining. This needs further investigation: scaling the features and the embeddings is one possible technique to try.

Another possible reason for unsupervised pretraining bringing no improvement might be in the use of the ReLU activations: Glorot et al. (2011) previously found that unsupervised pretraining does not help when the ReLU activation is used (with ReLU performing better than other activations even with unsupervised pretraining). The dropout regularisation could be yet another reason. As Srivastava et al. (2014)

state, "the stochastic nature of dropout might wipe out" the pretraining information.

All in all, our results suggest that pretrained word embeddings (1) do not always improve the performance of neural answer reranking systems, and (2) should only be used when suitable corpora are available, as pretraining on an out-of-domain corpus could decrease the performance drastically (as in the last row of Table 7.4).

7.4 Analysis

By conducting an error analysis on the YA dataset we were able to pinpoint the main causes of errors as follows:

1. Despite containing only *how* questions, the dataset contains a large amount of questions asking for an **opinion** or **advice** , e.g. *How should I do my eyes?*, *How do I look?* or *How do you tell your friend you're in love with him?* rather than **information**, e.g. *How do you make homemade lasagna?* and *how do you convert avi to mpg?* About half of the questions where the best system was still performing incorrectly were of the opinion-seeking nature. This is a problem for automatic answer reranking, since the nature of the question makes it very hard to predict the quality of the answers.

2. The choice of the best answer relies purely on the user. Inspection of the data reveals that these user-provided gold labels are not always reliable. In many cases the users tend to select as the best those answers that are most **sympathetic** (see Q1 in Table 7.5) or **funny** (see Q2 and Q3 in Table 7.5), rather than the ones providing more useful information.

In order to gain more insight into the reasons behind the errors on the YA data, we calculated average P@1 per category.[2] Figure 7.3 shows average P@1 of the LSTM-MLP-Discourse system versus the Random baseline for the most common categories. From this figure it is clear that the most challenging category for answer

[2]We first mapped the low-level categories provided in the dataset to the 26 high-level YA categories. We only consider categories that contained at least 100 questions.

(Q1) How does someone impress a person during a conversation that u are as good as an oxford/harvard grad.?
(Gold) i think you're chasing down the wrong path. but hell, what do i know?
(Prediction) There are two parts. Understanding your area well, and being creative. The understanding allows you the material for your own opinions to have heft and for you to analyse the opinions of others. After that, it's just good vocabulary which comes from reading a great deal and speaking with others. Like many other endeavors practice is what makes your performance improve.
(Q2) How to get my mom to stop smoking?
(Gold) Throw a glass of water on her every time she sparks one up
(Prediction) Never nag her. Instead politely insist on your right to stay free of all the risks associated with another person's smoking. For example, do not allow her to smoke inside the car, the house or anywhere near you (...)
(Q3) How do i hip hop dance??!?!?
(Gold) Basically, you shake what your mother gave you.
(Prediction) Listen to previous freestyle flows and battles by great artists (...) Understand the techniques those artists use to flow and battle (...)

Table 7.5: Example incorrect predictions of the system on the Yahoo! Answers dataset.

reranking is Family & Relationships. This category is also the most frequent in the dataset, with 494 out of 2500 questions belonging to it. Our system achieves about 4% lower P@1 on the questions from the Family & Relationships category than on the whole test set, while the random baseline performs as well as on the whole test set (the average number of answers per question in this category does not differ much from the dataset average). The low P@1 on this category is related to the reasons pointed out above: most questions in this category are of an opinion-seeking nature: *How do I know if my boyfriend really loves me?*, *How do I fix my relationship?*, *How do I find someone that loves me?*, making it hard to assess the quality of the answers.

The Ask Ubuntu dataset is rather different. In contrast to the YA dataset, which contains many subjective questions, most Ask Ubuntu questons relate to a complex technology and usually require deep domain knowledge to be answered.

(Q1) How do I add the kernel PPA? I can get Ubuntu mainline kernels from this kernel PPA - is there a way to add it to my repository list the same as regular Launchpad PPAs? (Gold) Warning : This answer is outdated. As of writing this warning (6.10.2013) the kernel-ppa used here is no longer updated. Please disregard this answer. `sudo apt-add-repository ppa:kernel-ppa/ppa sudo apt-get update sudo apt-get install PACKAGENAME` (Prediction) Since the kernel ppa is not really maintained anymore, here's a semi-automatic script: `https://github.com/medigeek/kmp-downloader`
(Q2) Which language is ubuntu-desktop mostly coded in? I heard it is Python (Gold) Poked around in Launchpad: ubuntu-desktop to and browsed the source for a few mins. It appears to be a mix of Python and shell scripts. (Prediction) I think the question referred to the language used to write the applications running on the default installation. It's hard to say which language is used the most, but i would guess C or C++. This is just a guess and since all languages are pretty equal in terms of outcome, it doesn't really matter.

Table 7.6: Example incorrect predictions of the system on the Ask Ubuntu dataset.

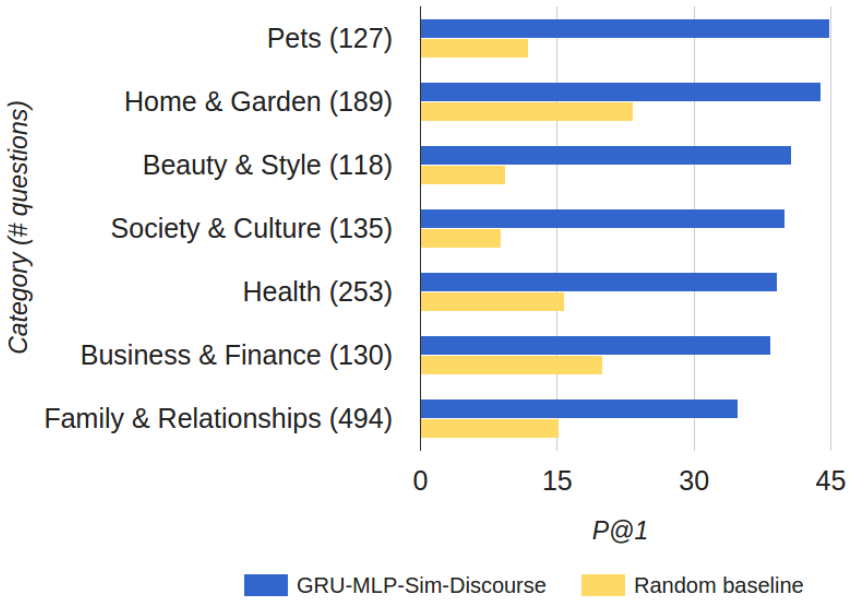

Figure 7.3: Average P@1 of the LSTM-MLP-Discourse versus the Random baseline on the test questions from most common YA categories.

Moreover, many questions and answers contain code, screenshots and links to external resources. Reliably reranking such answers based on textual information alone might be an unattainable goal. The technical complexity of the questions can give rise to ambiguity. For instance, in Q2 in Table 7.6 it is not clear if the question

	Train	Dev	Test
# questions	2669	500	700
# comments	17900	2440	3270
good	6651	818	1329
bad	8139	1209	1485
potentially useful	3110	413	456

Table 7.7: Details about Semeval 2016 Task 3 Subtask A data.

refers to the metapackage *ubuntu-desktop* or to ubuntu default packages in general. Another potential source of difficulty comes from the fact that the technologies being discussed on Ask Ubuntu change rapidly: some answers selected as best might be outdated (see Q1 in Table 7.6).

7.5 Experiments on SemEval Data

SemEval has been organising a shared task on community question answering since 2015: Task 3 in 2015 and 2016. This shared task contains several subtasks including Subtask A: Question-Comment Similarity. Given a question from a user forum and ten comments from the same forum thread, the goal is to rank these comments according to their relevance to the question. Every comment is labelled as *good*, *bad* or *potentially useful*. All *good* comments should be ranked before *bad* and *potentially useful*. A question can have several or no good answers. The dataset was created using the Qatar Living forum.[3] Table 7.7 shows the number of questions and comments in training, development and test sets. The official score of the shared task is Mean Average Precision (MAP).

We evaluate the models we described in Sections 6.1, 6.2 and 6.3 on this subtask's data. We tune the learning rate on the development set (see Table 7.8). All systems use dropout of 0.5 and weight decay of 10^{-5}. We set question length to 100 and answer length to 250.

We apply the three models with a learning rate of 0.001 to the test set. Table 7.9 presents the results obtained with the official scorer. The three systems provide

[3] `http://www.qatarliving.com/forum`

System/Encoder	Learning Rate	Dev MAP
CNN	0.001	**63.54**
CNN	0.01	61.36
LSTM	0.001	**65.67**
LSTM	0.01	45.77
MC-RCNN	0.001	**64.96**
MC-RCNN	0.01	62.91

Table 7.8: MAP on the Semeval 2016 Task 3 Subtask A development set.

System/Encoder	MAP	AvgRec	MRR	P	R	F1	Acc
CNN	74.85	85.52	82.12	71.77	70.55	52.45	60.16
LSTM	75.42	85.95	83.16	66.61	63.36	**64.94**	72.20
MC-RCNN	75.04	85.93	82.10	70.19	59.89	64.64	73.36
Chronological Baseline	59.53	72.60	67.83	-	-	-	-
Random Baseline	52.80	66.52	58.71	40.56	**74.57**	52.55	45.26
Filice et al. (2016)	**79.19**	**88.82**	**86.42**	**76.96**	55.30	64.36	**75.11**

Table 7.9: Performance on the Semeval 2016 Task 3 Subtask A test set. Calculated using the official scorer.

similar performance in terms of MAP and outperform the baselines by a substantial margin. The best performing system (Filice et al., 2016) at Subtask A achieved a MAP of 79.19, which is about 3.8% better than the LSTM-MLP system. The system of Filice et al. (2016) uses an SVM classifier with tree kernels and task-specific heuristic features that were previously proposed by Barrón-Cedeño et al. (2015). These heuristic features include the following binary features: whether a certain word (*yes*, *no*, *sure*, *can*, *neither*, *okay*, *sorry*, etc.) or a symbol (e.g. *?* or *@*) is present in a comment; whether the comment starts with *yes*; meta-information on whether the comment has been posted by the same user as the one who posted the question; whether the comment contains an acknowledgment (words containing *thank*). In contrast, neither of our systems uses any features. Perhaps the performance could be improved by incorporating them in a similar way to how the discourse features were incorporated.

7.6 Summary

In this chapter we continued to study the task of answer ranking. We suggested a way to combine our neural system with external features. We tested the discourse features produced by the Discourse Marker Model of Jansen et al. (2014). We estimated the impact of unsupervised pretraining on the neural model, and also compared the performance of the models that use character-level embeddings versus when they use word-level embeddings. We evaluated some of our model on the Semeval 2016 Task 3 data. Finally we provided some error analysis. Our main findings are:

- The use of character-level embeddings only is not as beneficial as when the word embeddings are used;
- Discourse features on their own provide a good baseline, and in most cases injecting them into a neural system improves the performance. They seem particularly helpful for longer texts;
- The CNN-based systems for encoding long texts benefit from the inclusion of the discourse information;
- Combining the MC-RCNN model with the discourse features does not improve its performance;
- Unsupervised pretraining of the word embeddings only slightly improves the performance of the neural systems, when no external features are used;
- Pretraining of word embeddings using an out-of-domain corpus can negatively affect the performance (when using the newswire embeddings with Ask Ubuntu data);
- On the YA dataset, the main source of error is the subjectivity of some of the questions and the choice of best answer;

- On the AU dataset, the main source of error is the complexity and the changing nature of the subject matter;

Chapter 8

Conclusion

In this thesis we explored CQAs as a source of labelled data for machine learning approaches to two different tasks, i.e. detection of semantically equivalent questions and answer ranking, with the latter being the main focus of this thesis. We primarily addressed the tasks using deep learning approaches. We explored the use of various architectures including feedforward, convolutional and recurrent neural networks and their combinations. Our experiments showed that this family of approaches provides good performance on the two tasks. In this chapter we first revisit the research questions we posed in Chapter 1, before outlining several directions for future work.

The first two research questions of this thesis concerned the limits of the neural approaches to the tasks of community question answering. In particular, the first question concerned the task of semantically equivalent question detection:

1. *Is it possible to predict semantically equivalent questions in community question answering websites using a deep learning system and relying on textual information only?*

   In Chapter 3 we approached this task by using a convolutional neural network. We presented experiments on a dataset of questions from the Ask Ubuntu community. We compared the system based on the CNN architecture with an SVM baseline and a system for duplicate detection based on shingling. Our exper-

iments showed that the neural approach outperforms the baselines by a large margin. Our experiments also showed that the use of word embeddings pretrained on an in-domain corpus, e.g. Ask Ubuntu data, is more beneficial than using embeddings pretrained on a general domain data, e.g. English Wikipedia, even if the size of the out-of-domain dataset is significantly larger. Finally, we investigated the impact of the training set size on the performance of the convolutional approach versus the SVM baseline. The experiments showed that the CNN-based system performs well even with limited amounts of training data, while the performance of the SVM baseline drops significantly when the amount of training data is reduced. In short, our findings show that it is possible to predict semantically equivalent questions using a neural system.

The second question concerns the limits of deep learning methods for the task of answer ranking in CQA:

2. *Can we rank answers to questions in community question answering websites without relying on handcrafted features?*

   In Chapter 5 we represented the questions and the answers using Paragraph Vector model (Le and Mikolov, 2014), and in Chapter 6 we explored other ways of representing questions and answers, including recurrent and convolutional neural networks. Passing these representations to a multilayer perceptron that is used for scoring them, we achieved better performance than the feature-based baselines of Jansen et al. (2014) and Fried et al. (2015) that provided previous state-of-the-art results on the same data.

The rest of the research questions concerned the neural approaches to the task of answer ranking. In particular, the third research question aimed to explore several neural architectures for the task of answer ranking including convolutional and recurrent neural networks:

3. *Which neural architectures are most suitable for encoding questions and answers in answer ranking?*

In Chapter 6 we compared the performance of the two most common variants of recurrent neural networks, Long Short Term Memory networks and Gated Recurrent Networks. Our experiments showed that they provide similar performance, with the LSTMs producing marginally better results. We also investigated the use of the attention mechanism of Bahdanau et al. (2014) for encoding questions and answers, and compared it to a simpler attention-like mechanism aiming to spot alignment between questions and answers. Our experiments found that incorporating the attention mechanism does not improve the ranking results, probably due to the absence of the actual alignment between questions and answers, unlike the case of machine translation, for which these mechanisms were designed. We also compared recurrent neural networks and convolutional neural networks for encoding questions and answers. We observed that when encoding short texts, CNNs provide performance which is competitive to RNNs. However, the CNNs are not suitable for encoding long documents. Finally, we combined the two architectures and proposed a novel neural architecture that we call Multi-Channel Convolutional Recurrent Neural Network. This architecture combines the benefits of recurrent and convolutional architectures. Experimental results showed that the novel architecture achieves state-of-the-art performance in answer ranking. In Chapter 7 we investigated the use of character-level instead of word-level word embeddings, and found that the representing the documents on character-level, i.e. using only character-level embeddings, usually provides worse performance than when word embeddings are used.

Our fourth research question concerned the possibility of combining traditional feature-based and the neural approaches:

4. *Can feature-based and neural approaches be successfully combined for the task of answer ranking? Do neural systems for answer ranking benefit from the inclusion of tried-and-tested features for this task?*

In Chapter 7 we enriched the neural system with the discourse features introduced by Jansen et al. (2014). Despite their simplicity, these features were part of the previous state-of-the-art system. On their own, they provide a good baseline. Our experiments showed that the neural approach benefits from the inclusion of these features. The improvement is especially notable on longer documents or when a convolutional neural network is used as an encoder.

Since we focus on CQAs, that do not restrict questions to any particular type, we investigate which questions are the most challenging from the point of view of automatic answer ranking:

5. *What kinds of questions pose the greatest challenge for the automatic answer ranking systems?*

   The error analysis of our systems presented in Chapter 7 showed that there are several types of questions that make automatic answer ranking and question answering challenging. Firstly, this is due to opinion-seeking questions common in social CQAs like Yahoo! Answers. Questions containing images and videos are yet another challenge for text-based systems like the ones we explored. We discuss in more detail about different types of questions in the next section.

8.1 Future Work

8.1.1 Creation of Gold Standards

In this study we assumed that CQAs provide natural annotation of the data, i.e. the community-provided best answers were treated as the gold best answers. Our experiments show that the machine learning approach can perform well on this task. However, an error analysis reveals that many of the errors are due to the subjective nature of the CQA data: first, a question could ask for an opinion, and second, the choice of the best answer depends on the author of the question. Hoogeveen

et al. (2016) performed an analysis of duplicate questions in the CQADupStack dataset (Hoogeveen et al., 2015) containing duplicate questions from various Stack Exchange communities. They have found that some duplicate questions lacked the necessary label. Similar findings were reported by Lei et al. (2016) for the Ask Ubuntu community.

Community-produced data is somewhat similar to the data produced by crowdsourcing, as the annotations are produced by non-experts. With the advance of such services as Amazon Mechanical Turk and Crowdflower, many research studies (Snow et al., 2008; Munro et al., 2010; Callison-Burch and Dredze, 2010) explored using non-expert annotations produced using crowdsourced data in various natural language processing tasks including word similarity prediction (Snow et al., 2008), recognising textual entailment (Munro et al., 2010; Snow et al., 2008), question generation (Heilman and Smith, 2010) and word sense disambiguation (Snow et al., 2008). Crowdsourced data is considered to be a *cheap* way to obtain data, in contrast to expert annotations. Nonetheless, it has been shown that it is possible to obtain data of a similar quality to expert annotations via crowdsourcing (Snow et al., 2008; Munro et al., 2010). Natural labelled data of CQAs that we explored in this thesis is a *free* source of data. Perhaps future research should consider verifying community labelled data via crowdsourcing and expert annotations. For instance, for weakly moderated communities like Yahoo! Answers question quality can be assessed, with non-information seeking questions being labelled. For Stack Exchange communities, the best answer labels can be verified by domain experts. This would help determining the upper bound for the CQA-based methods and creating more reliable gold standards for the CQA tasks.

8.1.2 Developing Interpretable Neural Architectures

We have shown that the neural approach in general performs well on the task of answer ranking. We have also empirically shown that some architectures and settings work better for the task than others, e.g. LSTMs outperform CNNs when both use

word-level information. Even though there is empirical evidence that some neural architectures perform well for certain tasks, many questions about their performance remain open and are being actively studied (Li et al. (2016), Collins et al. (2016) and Jozefowicz et al. (2015)). Even though deep learning aims to minimise the need for feature engineering, the uncertainty and uninterpretability of the neural systems cause the need for *architecture engineering*, i.e. searching for the best neural architecture, as we did in the chapters devoted to answer ranking. In many cases there are no clear answers to why one neural architecture performs well.

8.1.3 Developing Strategies for Hyperparameter Tuning

Typical neural architectures, like the ones we used in our experiments, have many hyperparameters and exploring all of them is unfeasible. For most models, we tuned only the learning rate, setting most hyperparameters based on our intuition, i.e. commonsense and experience, both ours and other researchers. A very thorough hyperparameter tuning for recurrent neural networks is presented by Collins et al. (2016), and yet they do not suggest an optimal strategy for hyperparameter settings. The main reasons are that the optimal sets of hyperparameters often depend on the task and the particular neural architecture. Even when using the same set of hyperparameters, performance may vary due to random initialisation.

8.1.4 Question Answering Evaluation

One of the main challenges of developing a live question answering system is the absence of means of automatic evaluation. This limits even the model selection process, which is important when building a neural system.

Several studies made attempts to address these issues by proposing metrics for automatic evaluation of question answering. For instance, Soricut and Brill (2004) used n-gram overlap to automatically evaluate answers to frequently asked questions. Lin and Demner-Fushman (2005) proposed a metric for automatic evalua-

tion of answers to definition questions called POURPRE. Definition questions were questions that are often expressed as *Tell me interesting things about X*, and can be rephrased as a set of factoid questions that are not known in advance, i.e. *Who is X? Where was he born? What is he famous for? What was his occupation?* etc. Their evaluation metric was inspired by nugget-based manual evaluation: a nugget is a fact for which an assessor can make a binary decision on whether it is included in the answer or not. The nuggets are divided into *vital* and *okay*. The POURPRE metric is based on the amount and the density of both vital and okay nuggets in the response. A nugget is considered to be in the answer if the normalised word overlap between the nugget and the answer exceeds a threshold. This measure showed a good correlation with the human judgements.

Keikha et al. (2014) suggested to evaluate web question answering using metrics for automatic evaluation of summarisation, i.e. ROUGE (Lin and Och, 2004), however, the correlation with human judgements was not very high. Moreover, the use of ROUGE to evaluate summarisation has been recently criticised, as it makes it impossible to achieve the perfect score and the relative perfect scores are unattainable by humans (Schluter, 2017). Several studies used the BLEU machine translation evaluation metric (Papineni et al., 2002) to evaluate answers to questions (Pérez et al., 2004; Noorbehbahani and Kardan, 2011). This approach, however, imposes restrictions on the types of questions that can be evaluated, i.e. there should only be one well defined correct answer. Future research should focus on developing methods for automatic evaluation of question answering systems.

8.1.5 Question Type Classification

In this thesis, we treated all questions in the same way, without looking closely at question types, assuming that CQA questions are of a non-factoid nature. In order to develop an end-to-end live question answering system, one has to make sure it is able to deal with different types of questions. Question type classification could be done as the first step, and different approaches could be taken depending

on the question type. For instance, opinion seeking questions and social questions would need to be addressed separately. Factoid questions and even some non-factoid questions may require access to a knowledge base in order to be answered.

We analysed questions of Text Retrieval Evaluation Conference (TREC) 2015 LiveQA track (Agichtein et al., 2015).[1] The TREC LiveQA track, unlike previous TREC QA tracks, involved answering real questions from Yahoo! Answers in real time. Each participant needed to submit a web service application that receives a question and responds with an answer. The questions, being sampled from a stream of real Yahoo! Answers questions, were much more diverse than in previous TREC QA tracks. The questions were not filtered by the organisers and included manner, opinion, advice and many other types of questions. Here we outline the main types of questions:

yes/no questions: questions that require either *yes* or *no* as an answer, e.g. *Are insects animals?* A binary classifier can be trained to answer these questions.

factoid questions: these are questions that can be answered with a named or a numerical entity, e.g. *Who is the president of Brazil?* or *When was the first smartphone invented?* We discussed these questions in Chapter 1. These questions can be answered using a knowledge base (Berant et al., 2013) or using an information retrieval based approach Kwok et al. (2000).

non-factoid information-seeking questions: these are information-seeking questions that cannot be answered with a named or numerical entity but can be answered with a paragraph, these are usually manner (*how)* and reason (*why*) questions, e.g. *How can I remove stains from my carpet?* A web-based approach (Wang and Nyberg, 2015) or a CQA-based approach combining candidate retrieval with answer ranking methods like the ones we discussed in Chapters 4-7 can be applied.

[1] `https://www.sites.google.com/site/trecliveqa2015/`

reasoning questions: these questions require commonsense reasoning to be answered (Rajpurkar et al., 2016; Weston et al., 2015b) For instance, *how much Indian am I if my grandmother on my father's side was 3/4 indian and my grandmother on my mother's side was half-indian?* To answer this kind of question, one would require not only commonsense reasoning but also knowledge of genetics and probability.

conversational questions: these are questions that do not seek information, but rather look for a conversation, e.g. *Do you like dogs?* or *I am so bored, please help me.* While these are legitimate questions to a chatbot system, these questions are usually not the focus of question answering systems, which are mostly oriented towards information seeking questions.

opinion seeking questions: Similarly to the previous category, these questions are not looking for information but rather need an opinion, e.g. *How should I do my hair today?* or *What is your opinion of Thailand?.*

question containing images/video: A few questions had images and video in them, e.g. *Is this burn or an infection? What is this?* `[image]` or *How old do i look considering this drawing?* `[image]`

Some questions fall into several of the above categories. For instance, some reasoning, opinion-seeking or multimodal questions are also *yes/no* questions. On CQAs, we can also observe many questions that are rather sets of related questions rather then one single question, e.g. *Why is the farmland divided in Très Riches Heures? Which class of feudal society is shown working the fields around the Manor? What visual elements assist you in identifying their class?*

8.1.6 End-to-End Live Question Answering

We have only addressed the tasks of classifying semantically equivalent questions and answer ranking. However, in order to build an end-to-end non-factoid questions

answering system, we need a question retrieval module. dos Santos et al. (2015) present an extension of the system we presented in Chapter 3, which uses the same convolutional neural network in combination with bag-of-words scoring. A similar system can be incorporated in a non-factoid question answering system.

Rücklé and Gurevych (2017) have recently presented a service architecture for an end-to-end non-factoid question answering system. The system can be built from any attention-based answer ranking model. They present a user interface visualising the attention on the selected answers, that allows the user to compare different attention-based answer selection mechanisms interactively.

An ideal question answering system should be able to answer all kinds of questions including factoid, non-factoid, reasoning questions and questions containing multi-modal information. One way to approach this task is to build various models, each of which would be able to answer a particular type of question, e.g. questions about images, yes/no questions, questions requiring domain knowledge of the Ubuntu operating system etc. Then, the question answering can be performed using a dialogue system, the natural language understanding component of which would convert the question into its semantic representation, and the dialogue manager component would be responsible for making a decision on whether to ask for clarifications or to answer the question and which one of the available models should be used in the latter case.

Developing a question answering system that would be able to deal with all sorts of questions is a very challenging task which goes to the core of real Natural Language Understanding and Artificial Intelligence and has many possible avenues for further research.

Bibliography

Agichtein, E., Carmel, D., Pelleg, D., Pinter, Y., and Harman, D. (2015). Overview of the TREC 2015 LiveQA Track. In *NIST Special Publication: SP 500-319: Proceedings of The Twenty-Fourth Text REtrieval Conference, TREC*.

Agichtein, E., Castillo, C., Donato, D., Gionis, A., and Mishne, G. (2008). Finding high-quality content in social media. In *Proceedings of the 2008 international conference on web search and data mining*, pages 183–194. ACM.

Amiri, H., Resnik, P., Boyd-Graber, J., and Daumé III, H. (2016). Learning text pair similarity with context-sensitive autoencoders. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1882–1892.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). VQA: visual question answering. In *Proceedings 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2425–2433.

Auli, M., Galley, M., Quirk, C., and Zweig, G. (2013). Joint language and translation modeling with recurrent neural networks. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1044–1054, Seattle, Washington, USA.

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2015). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Baroni, M., Dinu, G., and Kruszewski, G. (2014). Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland.

Barrón-Cedeño, A., Filice, S., Da San Martino, G., Joty, S., Màrquez, L., Nakov, P., and Moschitti, A. (2015). Thread-level information for comment classification in community question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 687–693, Beijing, China.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA.

Berger, A., Caruana, R., Cohn, D., Freitag, D., and Mittal, V. (2000). Bridging the lexical chasm: statistical approaches to answer-finding. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 192–199. ACM.

Berger, A. and Lafferty, J. (1999). Information retrieval as statistical translation. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 222–229, New York, NY, USA.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Bluche, T., Kermorvant, C., and Louradour, J. (2015). Where to apply dropout in recurrent neural networks for handwriting recognition? In *Proceedings of the 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 681–685.

Bogdanova, D., dos Santos, C., Barbosa, L., and Zadrozny, B. (2015). Detecting semantically equivalent questions in online user forums. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 123–131, Beijing, China.

Bogdanova, D. and Foster, J. (2016). This is how we do it: Answer reranking for open-domain how questions with paragraph vectors and minimal feature engineering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1290–1295, San Diego, California.

Bogdanova, D., Foster, J., Dzendzik, D., and Liu, Q. (2017). If you can't beat them join them: Handcrafted features complement neural nets for non-factoid answer reranking. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 121–131, Valencia, Spain.

Bonadiman, D., Uva, A., and Moschitti, A. (2017). Effective shared representations with multitask learning for community question answering. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 726–732, Valencia, Spain.

Broder, A. (1997). On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, pages 21–29, Washington, DC, USA.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Burke, R. D., Hammond, K. J., Kulyukin, V., Lytinen, S. L., Tomuro, N., and Schoenberg, S. (1997). Question answering from frequently asked question files: Experiences with the faq finder system. *AI magazine*, 18(2):57.

Cai, L., Zhou, G., Liu, K., and Zhao, J. (2011). Learning the latent topics for question retrieval in community qa. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 273–281.

Callison-Burch, C. and Dredze, M. (2010). Creating speech and language data with amazon's mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, CSLDAMT '10, pages 1–12, Stroudsburg, PA, USA.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. (2007). Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 129–136, New York, NY, USA.

Carlson, L., Marcu, D., and Okurowski, M. E. (2001). Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Proceedings of the Second SIGdial Workshop on Discourse and Dialogue - Volume 16*, SIGDIAL '01, pages 1–10, Stroudsburg, PA, USA. Association for Computational Linguistics.

Cartright, M.-A., White, R. W., and Horvitz, E. (2011). Intentions and attention in exploratory health search. In *Proceedings of the 34th international ACM SIGIR*

conference on Research and development in Information Retrieval, pages 65–74. ACM.

Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cheng, H., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., and Shah, H. (2016). Wide & deep learning for recommender systems. *CoRR*, abs/1606.07792.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar.

Chung, J., Cho, K., and Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1693–1703, Berlin, Germany.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Collins, J., Sohl-Dickstein, J., and Sussillo, D. (2016). Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Cong, G., Wang, L., Lin, C.-Y., Song, Y.-I., and Sun, Y. (2008). Finding question-answer pairs from online forums. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 467–474. ACM.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Das, A., Yenala, H., Chinnakotla, M., and Shrivastava, M. (2016). Together we stand: Siamese networks for similar question retrieval. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 378–387, Berlin, Germany.

De Andrade Barbosa, L., Bogdanova, D., Kormaksson, M., dos Santos, C., and Zadrozny, B. (2017). Machine learning and training a computer-implemented neural network to retrieve semantically equivalent questions using hybrid in-memory representations. US Patent 9,659,248.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE.

Denil, M., Demiraj, A., Kalchbrenner, N., Blunsom, P., and de Freitas, N. (2014). Modelling, visualising and summarising documents with a single convolutional neural network. *CoRR*, abs/1406.3830.

dos Santos, C., Barbosa, L., Bogdanova, D., and Zadrozny, B. (2015). Learning hybrid representations to retrieve semantically equivalent questions. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 694–699, Beijing, China.

dos Santos, C. and Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th Interna-*

tional Conference on Computational Linguistics: Technical Papers, pages 69–78, Dublin, Ireland.

dos Santos, C. N., Tan, M., Xiang, B., and Zhou, B. (2016). Attentive pooling networks. *CoRR*, abs/1602.03609.

dos Santos, C. N. and Zadrozny, B. (2014). Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages 1818–1826.

Drucker, H., Burges, C. J., Kaufman, L., Smola, A., and Vapnik, V. (1997). Support vector regression machines. *Advances in neural information processing systems*, 9:155–161.

Duan, H., Cao, Y., Lin, C.-Y., and Yu, Y. (2008). Searching questions by identifying question topic and question focus. In *ACL*, volume 8, pages 156–164.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., and Leyton-Brown, K. (2013). Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, pages 1–5.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660.

Fader, A., Zettlemoyer, L., and Etzioni, O. (2013). Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1608–1618, Sofia, Bulgaria.

Feng, V. W. and Hirst, G. (2012). Text-level discourse parsing with rich linguistic features. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 60–68. Association for Computational Linguistics.

Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., et al. (2010). Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79.

Filice, S., Croce, D., Moschitti, A., and Basili, R. (2016). Kelp at semeval-2016 task 3: Learning semantic relations between questions and answers. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1116–1123, San Diego, California.

Firat, O., Cho, K., and Bengio, Y. (2016). Multi-way, multilingual neural machine translation with a shared attention mechanism. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 866–875, San Diego, California.

Foster, J., Çetinoglu, Ö., Wagner, J., Le Roux, J., Hogan, S., Nivre, J., Hogan, D., and Van Genabith, J. (2011). # hardtoparse: Pos tagging and parsing the twitterverse. In *Proceedings of the Association for the Advancement of Artificial Intelligence 2011 Workshop On Analyzing Microtext*, pages 20–25.

Fried, D., Jansen, P., Hahn-Powell, G., Surdeanu, M., and Clark, P. (2015). Higher-order lexical semantic models for non-factoid answer reranking. *Transactions of the Association for Computational Linguistics*, 3:197–210.

Gabrilovich, E. and Markovitch, S. (2009). Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*, 34:443–498.

Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1019–1027.

Gers, F. A. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE.

Gillick, D., Brunk, C., Vinyals, O., and Subramanya, A. (2015). Multilingual language processing from bytes. *arxiv preprint 1512.00103*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 580–587.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Goldberg, D. E. and Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99.

Goldberg, Y. (2015). A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Graham, Y., Mathur, N., and Baldwin, T. (2014). Randomized significance tests in machine translation. In *Proceedings of the ACL 2014 Ninth Workshop on Statistical Machine Translation*, pages 266–274.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Gunning, R. (1952). The technique of clear writing. *McGraw-Hill, New York*.

Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Heilman, M. and Smith, N. A. (2010). Rating computer-generated questions with mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, CSLDAMT '10, pages 35–40, Stroudsburg, PA, USA.

Hieber, F. and Riezler, S. (2011). Improved answer ranking in social question-answering portals. In *Proceedings of the 3rd international workshop on Search and mining user-generated contents*, pages 19–26. ACM.

Higashinaka, R. and Isozaki, H. (2008). Corpus-based question answering for why-questions. In *Third International Joint Conference on Natural Language Processing, IJCNLP 2008, Hyderabad, India, January 7-12, 2008*, pages 418–425.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

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

Hoogeveen, D., Verspoor, K. M., and Baldwin, T. (2015). Cqadupstack: A benchmark data set for community question-answering research. In *Proceedings of the 20th Australasian Document Computing Symposium (ADCS)*, ADCS '15, pages 3:1–3:8, New York, NY, USA.

Hoogeveen, D., Verspoor, K. M., and Baldwin, T. (2016). Cqadupstack: Gold or silver? In *Proceedings of the SIGIR 2016 2nd Workshop on Web Question Answering (WebQA)*.

Hou, Y., Tan, C., Wang, X., Zhang, Y., Xu, J., and Chen, Q. (2015). Hitsz-icrc: Exploiting classification approach for answer selection in community question answering. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 196–202, Denver, Colorado.

Iyyer, M., Boyd-Graber, J., Claudino, L., Socher, R., and Daumé III, H. (2014). A neural network for factoid question answering over paragraphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 633–644, Doha, Qatar.

Jansen, P., Surdeanu, M., and Clark, P. (2014). Discourse complements lexical semantics for non-factoid answer reranking. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 977–986, Baltimore, Maryland.

Jenders, M., Krestel, R., and Naumann, F. (2016). Which answer is best?: Predicting accepted answers in mooc forums. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 679–684. International World Wide Web Conferences Steering Committee.

Jeon, J., Croft, W. B., and Lee, J. H. (2005). Finding similar questions in large question and answer archives. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 84–90. ACM.

Joachims, T. (2006). Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM.

Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 2342–2350.

Jurafsky, D. and Martin, J. H. (2014). *Speech and language processing*, volume 3. Pearson London.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732.

Katz, B. (1997). Annotating the world wide web using natural language. In *Computer-Assisted Information Searching on Internet*, pages 136–155. Le Centre de Hautes Etudes Internationales d'Informatique Documentaire.

Keikha, M., Park, J. H., Croft, W. B., and Sanderson, M. (2014). Retrieving passages and finding answers. In *Proceedings of the 2014 Australasian Document Computing Symposium*, page 81. ACM.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2741–2749. AAAI Press.

Kingma, D. and Adam, J. B. (2015). A method for stochastic optimisation. In *Proceedings of 5th International Conference on Learning Representations, ICLR 2015*.

Kolomiyets, O. and Moens, M.-F. (2011). A survey on question answering technology from an information retrieval perspective. *Information Sciences*, 181(24):5412–5434.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., and Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*, pages 1378–1387.

Kwok, K. L., Grunfeld, L., Dinstl, N., and Chan, M. (2000). Trec-9 cross language, web and question-answering track experiments using pircs. In *Proceedings of the Text Retrieval Conference (TREC-9*, pages 26–35.

Lau, J. H. and Baldwin, T. (2016). An empirical evaluation of doc2vec with practical insights into document embedding generation. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 78–86, Berlin, Germany.

Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1188–1196.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lei, T., Joshi, H., Barzilay, R., Jaakkola, T., Tymoshenko, K., Moschitti, A., and Màrquez, L. (2016). Semi-supervised question retrieval with gated convolutions. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1279–1289, San Diego, California.

Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26. ACM.

Li, J., Chen, X., Hovy, E., and Jurafsky, D. (2016). Visualizing and understanding neural models in nlp. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 681–691, San Diego, California.

Li, X. and Roth, D. (2006). Learning question classifiers: the role of semantic information. *Natural Language Engineering*, 12(03):229–249.

Liang, P. (2013). Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*.

Lin, C.-Y. and Och, F. J. (2004). Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 605. Association for Computational Linguistics.

Lin, J. (2007). An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems (TOIS)*, 25(2):6.

Lin, J. and Demner-Fushman, D. (2005). Automatically evaluating answers to definition questions. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 931–938. Association for Computational Linguistics.

Liu, Y., Bian, J., and Agichtein, E. (2008). Predicting information seeker satisfaction in community question answering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 483–490. ACM.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal.

Manning, C. (2017). Lecture 1 — natural language processing with deep learning. Stanford University School of Engineering University Lecture, `https://youtu.be/OQQ-W`$_6$`3UgQ?t=21m00s`.

Marchionini, G. (2006). Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46.

Marcu, D. (1997). *The Rhetorical Parsing, Summarization, and Generation of Natural Language Texts*. Ph.D. thesis.

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

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Mihaylova, T., Gencheva, P., Boyanov, M., Yovcheva, I., Mihaylov, T., Hardalov, M., Kiprov, Y., Balchev, D., Koychev, I., Nakov, P., et al. (2016). Super team at semeval-2016 task 3: Building a feature-rich system for community question answering. *Proceedings of SemEval*, pages 836–843.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proceedings of Interspeech 2010*, volume 2, page 3.

Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Mnih, V., Heess, N., Graves, A., et al. (2014). Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212.

Monz, C. (2004). Minimal span weighting retrieval for question answering. In *Proceedings of the SIGIR Workshop on Information Retrieval for Question Answering*, volume 2.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Proceedings of the Tenth International Conference on Artificial Intelligence and Statistics*, volume 5, pages 246–252.

Moschitti, A. (2006). Efficient convolution kernels for dependency and constituent syntactic trees. In *European Conference on Machine Learning*, pages 318–329. Springer.

Munro, R., Bethard, S., Kuperman, V., Lai, V. T., Melnick, R., Potts, C., Schnoebelen, T., and Tily, H. (2010). Crowdsourcing and language studies: the new generation of linguistic data. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 122–130, Los Angeles.

Muthmann, K. and Petrova, A. (2014). An Automatic Approach for Identifying Topical Near-Duplicate Relations between Questions from Social Media Q/A . In *Proceeding of WSDM 2014 Workshop: Web-Scale Classification: Classifying Big Data from the Web.*

Nakov, P., Màrquez, L., and Guzmán, F. (2016a). It takes three to tango: Triangulation approach to answer ranking in community question answering. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1586–1597, Austin, Texas.

Nakov, P., Màrquez, L., Moschitti, A., Magdy, W., Mubarak, H., Freihat, A. A., Glass, J., and Randeree, B. (2016b). Semeval-2016 task 3: Community question answering. *Proceedings of SemEval*, 16.

Nicosia, M., Filice, S., Barrón-Cedeño, A., Saleh, I., Mubarak, H., Gao, W., Nakov, P., Da San Martino, G., Moschitti, A., Darwish, K., Màrquez, L., Joty, S., and Magdy, W. (2015). Qcri: Answer selection for community question answering - experiments for arabic and english. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 203–209, Denver, Colorado.

Noorbehbahani, F. and Kardan, A. A. (2011). The automatic assessment of free text answers using a modified bleu algorithm. *Computers & Education*, 56(2):337–345.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Palmer, M., Gildea, D., and Kingsbury, P. (2005). The proposition bank: a corpus annotated with semantic roles. *Computational Linguistics Journal*, 31(1).

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Paşca, M. (2003). Open-domain question answering from large text collections.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318.

Pedersen, T., Patwardhan, S., and Michelizzi, J. (2004). Wordnet:: Similarity: measuring the relatedness of concepts. In *Demonstration papers at HLT-NAACL 2004*, pages 38–41. Association for Computational Linguistics.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar.

Pérez, D., Alfonseca, E., and Rodríguez, P. (2004). Application of the bleu method for evaluating free-text answers in an e-learning environment. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*, pages 1351–1354.

Pham, V., Bluche, T., Kermorvant, C., and Louradour, J. (2014). Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 285–290. IEEE.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. `http://is.muni.cz/publication/884893/en`.

Riezler, S., Vasserman, A., Tsochantaridis, I., Mittal, V., and Liu, Y. (2007). Statistical machine translation for query expansion in answer retrieval. In *Annual Meeting-Association For Computational Linguistics*, volume 45, page 464.

Ritter, A., Clark, S., Mausam, and Etzioni, O. (2011). Named entity recognition in tweets: An experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1524–1534, Stroudsburg, PA, USA.

Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M., and Gatford, M. (1994). Okapi at trec-3. In *Proceedings of the 3rd Text REtrieval Conference*, pages 109–126.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Rücklé, A. and Gurevych, I. (2017). End-to-end non-factoid question answering with an interactive visualization of neural attention weights. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL 2017)*, page (to appear).

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representation by back propagation. *Parallel distributed processing: exploration in the microstructure of cognition*, 1.

Salton, G. and McGill, M. J. (1986). *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

Schluter, N. (2017). The limits of automatic summarisation according to rouge. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 41–45, Valencia, Spain.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Severyn, A. and Moschitti, A. (2015a). Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 373–382. ACM.

Severyn, A. and Moschitti, A. (2015b). Unitn: Training deep convolutional neural network for twitter sentiment classification. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), Association for Computational Linguistics, Denver, Colorado*, pages 464–469.

Sharp, R., Jansen, P., Surdeanu, M., and Clark, P. (2015). Spinning straw into gold: Using free text to train monolingual alignment models for non-factoid question answering. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 231–237, Denver, Colorado.

Shen, L. and Joshi, A. K. (2005). Ranking and reranking with perceptron. *Machine Learning*, 60(1-3):73–96.

Shen, Y., Rong, W., Sun, Z., Ouyang, Y., and Xiong, Z. (2015). Question/answer matching for cqa system via combining lexical and sequential information. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 275–281.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sneiders, E. (2002). Automated question answering using question templates that cover the conceptual model of the database. In *NLDB*, volume 2, pages 235–239. Springer.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

Snow, R., O'Connor, B., Jurafsky, D., and Ng, A. (2008). Cheap and fast – but is it good? evaluating non-expert annotations for natural language tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 254–263, Honolulu, Hawaii.

Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136.

Soricut, R. and Brill, E. (2004). Automatic question answering: Beyond the factoid. In *HLT-NAACL 2004: Main Proceedings*, pages 57–64, Boston, Massachusetts, USA.

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

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Surdeanu, M. and Ciaramita, M. (2007). Robust information extraction with perceptrons. In *Proceedings of the NIST 2007 Automatic Content Extraction Workshop (ACE07)*.

Surdeanu, M., Ciaramita, M., and Zaragoza, H. (2008). Learning to rank answers on large online QA collections. In *Proceedings of ACL-08: HLT*, pages 719–727, Columbus, Ohio.

Surdeanu, M., Ciaramita, M., and Zaragoza, H. (2011). Learning to rank answers to non-factoid questions from web collections. *Comput. Linguist.*, 37(2):351–383.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Tan, M., Xiang, B., and Zhou, B. (2015). Lstm-based deep learning models for non-factoid answer selection. *CoRR*, abs/1511.04108.

Tang, D., Qin, B., and Liu, T. (2015a). Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*, pages 1422–1432.

Tang, D., Qin, B., and Liu, T. (2015b). Learning semantic representations of users and products for document level sentiment classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1014–1023, Beijing, China.

Toba, H., Ming, Z.-Y., Adriani, M., and Chua, T.-S. (2014). Discovering high quality answers in community question answering archives using a hierarchy of classifiers. *Information Sciences*, 261:101–115.

Tran, Q. H., Tran, V., Vu, T., Nguyen, M., and Bao Pham, S. (2015). Jaist: Combining multiple features for answer selection in community question answering. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 215–219, Denver, Colorado.

Tymoshenko, K., Bonadiman, D., and Moschitti, A. (2016a). Convolutional neural networks vs. convolution kernels: Feature engineering for answer sentence reranking. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1268–1278, San Diego, California.

Tymoshenko, K., Bonadiman, D., and Moschitti, A. (2016b). Learning to rank non-factoid answers: Comment selection in web forums. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2049–2052. ACM.

Verberne, S., Boves, L., Oostdijk, N., and Coppen, P.-A. (2007). Discourse-based answering of why-questions. *Traitement Automatique des Langues, Discours et document: traitements automatiques*, 47(2):21–41.

Verberne, S., Boves, L., Oostdijk, N., and Coppen, P.-A. (2010). What is not in the bag of words for why-qa? *Computational Linguistics*, 36(2):229–245.

Verberne, S., van Halteren, H., Theijssen, D., Raaijmakers, S., and Boves, L. (2011). Learning to rank for why-question answering. *Information Retrieval*, 14(2):107–132.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408.

Voorhees, E. M. and Tice, D. M. (1999). The trec-8 question answering track evaluation. In *TREC*, volume 1999, page 82.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(3):328–339.

Wang, B., Wang, X., Sun, C., Liu, B., and Sun, L. (2010). Modeling semantic relevance for question-answer pairs in web social communities. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1230–1238. Association for Computational Linguistics.

Wang, D. and Nyberg, E. (2015). CMU OAQA at TREC 2015 liveqa: Discovering the right answer with clues. In *Proceedings of The Twenty-Fourth Text REtrieval Conference, TREC 2015, Gaithersburg, Maryland, USA, November 17-20, 2015*.

Wang, K., Ming, Z., and Chua, T.-S. (2009). A syntactic tree matching approach to finding similar questions in community-based qa services. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 187–194, New York, NY, USA.

Wang, M., Smith, N. A., and Mitamura, T. (2007). What is the Jeopardy model? a quasi-synchronous grammar for QA. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 22–32, Prague, Czech Republic.

Wang, N. and Yeung, D.-Y. (2013). Learning a Deep Compact Image Representation for Visual Tracking. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, page 809–817.

Wang, W., Yang, N., Wei, F., Chang, B., and Zhou, M. (2017). Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 189–198, Vancouver, Canada.

Wen, T.-H., Gasic, M., Mrkšić, N., Su, P.-H., Vandyke, D., and Young, S. (2015). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721, Lisbon, Portugal.

Weston, J., Bordes, A., Chopra, S., and Mikolov, T. (2015a). Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698.

Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., and Mikolov, T. (2015b). Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

Wu, Q., Teney, D., Wang, P., Shen, C., Dick, A. R., and van den Hengel, A. (2016). Visual question answering: A survey of methods and datasets. *CoRR*, abs/1607.05910.

Wu, Y., Zhang, Q., and Huang, X. (2011). Efficient Near-Duplicate Detection for Q&A Forum. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1001–1009, Chiang Mai, Thailand. Asian Federation of Natural Language Processing.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, volume 14, pages 77–81.

Xue, X., Jeon, J., and Croft, W. B. (2008). Retrieval models for question and answer archives. In *Proceedings of the 31st Annual International ACM SIGIR*

Conference on Research and Development in Information Retrieval, SIGIR '08, pages 475–482, New York, NY, USA.

Yang, L., Ai, Q., Spina, D., Chen, R.-C., Pang, L., Croft, W. B., Guo, J., and Scholer, F. (2016). Beyond factoid qa: effective methods for non-factoid answer sentence retrieval. In *European Conference on Information Retrieval*, pages 115–128. Springer.

Yi, L., Wang, J., and Lan, M. (2015). Ecnu: Using multiple sources of cqa-based information for answers selection and yes/no response inference. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 236–241, Denver, Colorado.

Yu, L., Hermann, K. M., Blunsom, P., and Pulman, S. (2014). Deep learning for answer sentence selection. *arXiv preprint arXiv:1412.1632*.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *CoRR*, abs/1409.2329.

Zhang, K., Wu, W., Wu, H., Li, Z., and Zhou, M. (2014). Question retrieval with high quality answers in community question answering. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 371–380, New York, NY, USA.

Zhang, P., Goyal, Y., Summers-Stay, D., Batra, D., and Parikh, D. (2016). Yin and Yang: Balancing and answering binary visual questions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5014–5022.

Zhang, W., Ming, Z., Zhang, Y., Nie, L., Liu, T., and Chua, T.-S. (2012). The use of dependency relation graph to enhance the term weighting in question retrieval. In *Proceedings of COLING 2012*, pages 3105–3120, Mumbai, India.

Zhou, G., Cai, L., Zhao, J., and Liu, K. (2011). Phrase-based translation model for question retrieval in community question answer archives. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 653–662, Stroudsburg, PA, USA.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.