Text Understanding for Computers

Tom Kenter

Text Understanding for Computers

Tom Kenter

Text Understanding for Computers

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in
de Agnietenkapel
op vrijdag 15 december 2017, te 12:00 uur

door

Tom Kenter

geboren te Wormer, Nederland

Promotiecommissie

Promotor:		
	Prof. dr. M. de Rijke	Universiteit van Amsterdam
Co-promotor:		
	Prof. dr. J. van Eijnatten	Universiteit Utrecht
Overige leden:		
	prof. dr. K. Balog	University of Stavanger
	prof. dr. A. van den Bosch	Radboud Universiteit, Meertens Instituut
	prof. dr. F.M.G. de Jong	Universiteit Utrecht
	dr. E. Kanoulas	Universiteit van Amsterdam
	dr. C. Monz	Universiteit van Amsterdam
	prof. dr. K. Sima'an	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

The research was supported by the Netherlands Organisation for Scientific Research (NWO) under project number HOR-11-10.

Cover photograph and design by Tom Kenter
Printed by Off Page, Amsterdam

ISBN: 978-94-6182-859-0

Dankwoord

Ten allereerste wil ik Josje, mijn vriendin, bedanken. Hoewel ik de vier jaar als promovendus alles bij elkaar als geweldig heb ervaren waren niet alle momenten even fantastisch, en ik ben bang dat Josje meer heeft meegemaakt van de mindere momenten van het promoveren dan van de leuke kanten (congresdeelname op exotische locaties, stages in het buitenland, etcetera). Josje, ik ben je dankbaar voor veel meer dan ik hier kan opnoemen. Dat was al zo voordat ik ging promoveren, dat was zo in de afgelopen jaren, en ik hoop dat dat nog heel lang zo zal blijven.

Ten tweede wil ik mijn vader bedanken. Ik had bijna mijn studie Alfa Informatica niet afgemaakt en het spreekt voor zich dat ik zonder dat diploma nooit aan een promotie had kunnen beginnen. Het is aan niets anders dan zijn goede zorgen te danken dat ik dat wel heb gedaan.

De eerste die me deed realiseren dat een promotie misschien iets voor me was is Folgert Karsdorp. Hij heeft daar niets bewusts voor gedaan maar toen hij weg ging bij het INL, waar we op dat moment allebei werkten, om te gaan promoveren merkte ik dat ik jaloers op hem was. En dat heeft me ertoe doen besluiten om bij de UvA te gaan solliciteren. Dank daarvoor! De tweede die me op weg hielp was David Graus. Hij begon aan zijn promotie bij ILPS terwijl ik wetenschappelijk programmeur in dezelfde groep was. Door met hem, Edgar Meij en Marc Bron te discussiëren over, en te werken aan een bijdrage voor een congres besefte ik dat ik hun baan als onderzoekers veel leuker vond dan mijn eigen baan als programmeur.

Here is a special shout-out to all people at ILPS. ILPS is a great group, and I really enjoyed getting to know and collaborate with all the smart people in it. The first full paper I ever wrote was with Krisztian Balog, a former ILPS'er, whom I am proud to have as a member of the committee for this thesis. Another co-author I learnt a lot from, and always enjoy talking to, is Alexey Borisov. I had a lot of fun times with everyone: Chuan Wu, Fei Cai, Ridho Reinanda, Harrie Oosterhuis, Marlies van der Wees, Daan Odijk, Anne Schuth, Katya Garmash, Marzieh Fadaee, Hamid Reza Ghader, Mostafa Dehghani, Hosein Azarbonyad, Bob van de Velde, Christophe Van Gysel, Julia Kiseleva, Petra Best of course, and everyone else.

I learnt a lot during the internships I did at Google Research. I want to thank everyone in the Wavii team and in the Expander team, in particular Dana Movshovitz-Attias, Daniel Hewlett, and Llion Jones (who taught me more in two days of pair programming than I could have learnt in a month reading papers).

Zeer veel dank ook aan Ruud Custers, die zich reeds voordat ik officieel was begonnen met promoveren had opgeworpen als paranimf!

Mijn promotie heeft plaatsgevonden in het kader van het Translantis project en ik wil graag alle projectgenoten bedanken.

Van alle mensen die bij hebben gedragen aan alle ervaringen van de afgelopen jaren, ben ik mijn promotor, Maarten de Rijke, het meest dankbaar. Ik heb het gevoel dat ik een nieuwe wereld, de academische wereld, heb leren kennen. Het promoveren heeft me op allerlei plekken gebracht, van congressen in Moskou, Dublin, Melbourne, Washington en Tokyo tot de stages bij Google in Californië. Dat was zonder dat Maarten mij het voordeel van de twijfel had gegeven door me aan te nemen nooit gebeurd en daar ben

ik hem bijzonder dankbaar voor. Ik mis de gesprekken die we wekelijks hadden. Ik heb erg veel van hem geleerd waar ik denk de rest van mijn leven nog veel aan zal hebben. Het is bijzonder dat een groep met zoveel promovendi zo goed functioneert en dat is met name aan Maarten te danken, die veel tijd voor zijn studenten vrijmaakt en nooit hun persoonlijke ontwikkeling uit het oog verliest. Maarten, je bent een bijzonder persoon, ik beschouw het als een voorrecht dat ik bij je heb kunnen promoveren, het is een groot genoegen met je samen te werken, en ik ben je uitermate dankbaar voor alles.

Tom Kenter
29 oktober 2017

Contents

1 Introduction

A long-standing challenge for computers communicating with humans is to pass the Turing test [148], i.e., to communicate in such a way that it is impossible for humans to determine whether they are talking to a computer or another human being. Interestingly, while this challenge has not been met yet, the bar has been raised while trying. Where the original Turing test was set in two adjacent rooms in a laboratory environment, the goal in human-computer interaction nowadays, as pursued by digital assistants like the Google Assistant,[1] Siri,[2] Cortana,[3] and Alexa,[4] is for the computer to be omni-present, omniscient and omnicompetent. That is, it should always be around to help, and should always be connected to the rest of the world. It should know everything, be aware of the latest news, speak all languages, and have all of Wikipedia memorized. Lastly, it should be able to perform actions for you: book concert tickets, transfer money, call your friends, manage your calendar and operate your fridge.

The field of *natural language understanding* — which studies automatic means of capturing the semantics of textual content — plays a central part in this long-term goal of artificial intelligence research. The task encompasses many subtasks, including machine translation [10, 94, 95, 145], question answering [42, 65, 159, 164, 165], document classification [28, 29, 88, 163, 170], summarization [112, 131], and text matching [70, 79, 84, 118, 119]. The span of associated techniques involved being this wide, research in natural language understanding contributes to research in other disciplines as well. In digital humanities research, where questions motivated from a humanities perspective, are answered using text analysis, natural language understanding plays an important role [55, 57, 60, 66, 71, 83, 89, 114, 115, 154, 160]. For example in the Translantis project[5] humanities scholars aim to answer questions about the emergence of the United States as a reference culture in the Netherlands by studying Dutch public discourse, in the form of digitized newspaper articles. Many of the subtasks of natural language understanding listed above, including machine translation, text matching and document classification are immediately relevant in this context.

Recent advances in AI research have changed the way computers read and comprehend documents. Where previous approaches relied on word counts and human input

[1] https://assistant.google.com/
[2] https://www.apple.com/ios/siri/
[3] https://www.microsoft.com/en-us/windows/cortana
[4] https://alexa.amazon.com
[5] http://translantis.wp.hum.uu.nl/

in the form of heuristics and sometimes hand-made rules [72, 79, 116, 134], modern day machine reading algorithms advance towards capturing the semantics of textual content from scratch, without the need for explicit rules or heuristics [31, 64, 65, 86]. In this thesis, contributions are made at both sides of this spectrum. Additionally, natural language understanding can itself be understood at different levels. We make contributions to automatic understanding of text at the level of words, short texts, and full documents.

Understanding texts at a word level, means understanding how words relate to each other semantically. For example, do two words or phrases mean approximately the same thing? Does a particular word still mean the same thing it used to, say, 50 years ago? Or, as is the question central to the first part of this thesis, can we automatically detect which words people used in different periods in time to refer to a particular concept?

When word-level semantics are understood to a sufficient degree, an attempt can be made at capturing the meaning of short pieces of text, such as sentences. The question we ask ourselves in the second part of this thesis is: can we automatically determine, from the word-level up, whether two sentences have a similar meaning?

Finally, in the third part of this thesis, document-level text understanding is the focus of our interest. In particular, we study multiple approaches to the reading comprehension task, where a computer reads a document and answers questions about it.

To sum up, the contributions made in this thesis to the field of natural language understanding are presented in three main parts:

Part I: Words Monitoring changes in word usage over time. Can we keep track, in an automated way, of the words people use to refer to a particular concept, in different periods in time?

Part II: Short texts Semantic similarity of sentences. How do we optimally use word representations to capture semantics at the level of short texts?

Part III: Documents Reading comprehension for computers. Can computers read and comprehend texts, so they can answer questions about them?

1.1 Research outline and questions

Part I: Words

Word-level semantics have been studied for many years. Digitized dictionaries have been used for decades [74, 97] and the creation of WordNet [121] — a network of words, grouped together in sets of synonyms that can be connected by, e.g., hypernym, hyponym or antonym relations — in 1995 spawned many initiatives for incorporating word-level semantics [13, 44, 45, 117, 134], leveraging the explicitly defined relations in WordNet. A more implicit way of capturing semantic information is provided by distributional semantics, which relies on the distributional hypothesis [118, 119, 124]. This hypothesis states that words occurring in similar contexts tend to have similar meanings [59], immortalized by Firth in 1957 as “You shall know a word by the company it keeps” [46].

One straightforward way of implementing contextual information per word is to represent words as co-occurrence vectors [17, 133]. A co-occurrence vector has as many dimensions as there are words in the corpus under study, where each value indicates the number of times the corresponding word co-occurred with the word the vector is for. Due to the large number of word types typically used in real-world corpora, this method usually yields high-dimensional vectors which cause computation to be inefficient. To avoid high computational costs, many ways of reducing the dimensionality have been proposed [21, 37, 40]. Recently, word2vec [118, 119] has become a popular method for obtaining low-dimensional word vectors, based on their context. It provides a fast and robust way of obtaining vectors which embed the words they correspond to in a, so-called, semantic space. One assumption corpus-based distributional semantic methods make is that the meaning words have, as represented by the relative positions of their vectors in the semantic space, is static over time. That is, the corpus the vectors are learnt from is taken as a bag of documents, or contexts, and no temporal order is taken into account. However, in real life, word meanings change over time. What is more, words used to refer to concepts change, while the underlying concepts stay relatively stable [101]. For example, personal portable audio players have been popular since the late seventies, when the *walkman* was invented. The introduction of the cd gave rise to *discmans* in the late eighties, and after the turn of the century the *iPod* took over. In Chapter 4, we address the task of monitoring shifts in word usage over time. That is, given a small seed set of words (e.g., *walkman*), we are interested in monitoring which terms are used over time to refer to the underlying concept denoted by the seed words (such as *discman* and *iPod*). Specifically, we aim to answer the following research question:

RQ1 Given a corpus of time-stamped documents, a point in time and a small set of seed terms used to denote a concept in that corpus at the time specified, can we infer from the corpus which terms are used in adjacent periods in time to denote the same underlying concept?

To solve RQ1 we propose an algorithm for monitoring shifts in vocabulary over time, given a small set of seed terms. We use distributional semantic methods to infer a series of semantic spaces over time from a large body of time-stamped, but otherwise unstructured textual documents. We construct semantic networks of terms based on their representations in the semantic spaces and use graph-based measures to calculate saliency of terms. Based on the graph-based measures we produce ranked lists of terms that represent the concept underlying the initial seed terms over time as final output.

Part II: Short texts

Word embeddings, mentioned above, prove to be a robust way of representing word-level semantics. It is not obvious, however, how they can be leveraged to capture the meaning of longer text segments like sentences.

Determining semantic similarity between texts [2–5] is important in many tasks such as finding similar queries issued to a search engine [24, 157], suggesting alternative queries [142], summarizing texts [6] and finding images [34, 164]. Approaches have been suggested based on lexical matching [13, 72, 75], handcrafted patterns [113, 134],

syntactic parse trees [140], and external sources of structured semantic knowledge [27, 48]. However, lexical features, like string matching, do not capture semantic similarity beyond a trivial level. Furthermore, generic external sources of structured semantic knowledge cannot be assumed to be available for low-resource languages. For example, WordNet is not available in every language, and the number of entries in the available WordNets vary wildly across languages,[6] a state of affairs that applies to Wikipedia[7] too. Additionally, handcrafted patterns are expensive to expand or create for previously unforeseen domains. Finally, approaches depending on parse trees are restricted to syntactically well-formed texts, typically of one sentence in length. Therefore, in this part, we discuss two approaches to matching short texts semantically, based on distributional semantics only. First, in Chapter 5, we treat word vectors as a given source of semantic information, and we match two short text fragments based on the embeddings of the words they contain. Second, in Chapter 6, we present the *Siamese Continuous Bag of Words* (Siamese CBOW) model, a neural network for efficient estimation of word embeddings that yield high-quality sentence embeddings.

In Chapter 5, we investigate whether determining short text similarity is possible using only semantic features — where by *semantic* we mean, pertaining to a representation of meaning — rather than relying on similarity between lexical or syntactic representations. We propose to go from word-level to text-level semantics by combining insights from methods based on external sources of semantic knowledge with word embeddings. In particular, we study the following research question:

RQ2 How can pre-trained word embeddings be used to calculate similarity between short texts, without relying on linguistic structure?

A novel feature of our approach is that an arbitrary number of word embedding sets is incorporated in one method. Moreover, it is not necessary for the embeddings in the different sets to be trained on the same corpus, cover the same vocabulary or even have the same dimensions. We derive multiple types of meta-features from the comparison of the word vectors for short text pairs, and from the vector means of their respective word embeddings. The features representing labelled short text pairs are used to train a supervised learning algorithm. We use the trained model at testing time to predict the semantic similarity of new, unlabelled pairs of short texts.

In Chapter 6 we optimize word embeddings directly. Averaging the embeddings of words in a sentence has proven to be a successful and efficient way of obtaining sentence embeddings. However, word embeddings trained with the methods currently available are not optimized for the task of sentence representation, and, thus, are likely to be suboptimal. Therefore, in this chapter, we aim to solve the following research question:

RQ3 Is it beneficial for word embeddings to be optimized for the task of being averaged to represent short texts?

[6] See `http://globalwordnet.org/wordnets-in-the-world/` for an overview of WordNets in different languages.

[7] See `https://en.wikipedia.org/wiki/List_of_Wikipedias`.

This problem is addressed by training word embeddings directly for the purpose of being averaged. We present a neural network architecture that learns word embeddings by predicting, from a sentence representation, its surrounding sentences.

Part III: Documents

In this part of the thesis, we focus on the task of machine reading, where a computer reads a document and answers questions about it. In this setting no use is made of external sources of knowledge, like knowledge bases. Rather, the answers to all questions should be inferred from reading the document provided.

Recently sequence-to-sequence architectures [145] have been proposed in a machine translation setting. Such a neural network processes a sequence of input word embeddings, and outputs a sequence of word embeddings. Apart from being successful in a machine translation setting, sequence-to-sequence learning has proven to be a powerful and successful paradigm for many other tasks, where both the input and output can be treated as sequences of symbols [64, 65, 131, 135]. In the third part of this thesis, we use sequence-to-sequence models for a machine reading task. A machine reads a sequence of input symbols, a document and a question, and outputs a sequence of symbols, thereby answering the question. In languages like English, using words as input symbols is a natural choice. Apart from slight morphological variations due to inflections based on, e.g., number (singular/plural), the same word usually has the same appearance, which makes it possible for a word-level model to capture meaningful patterns in the English words it observes. In many other languages, however, words are subject to much richer inflections than in English (e.g., Czech, Russian) or to highly productive prefix- and suffix attachments (e.g., Turkish). Due to these morphological phenomena, words are much less uniform compared to English. As a result, words are less suitable as units of input/output for a sequence-to-sequence model. Characters, or even more fundamental, bytes, may be better as input symbols in these languages.

In Chapter 7, we study a sequence-to-sequence architecture for solving a machine reading task in English by reading and outputting sequences of words. In particular, memory networks are treated. Memory networks have proven to be an effective architecture in machine reading tasks [54, 99, 120, 144, 158, 164], while sequence-to-sequence models have proven to be beneficial in many natural language processing settings. However, by using raw embeddings as input representations, many existing memory networks fail to make optimal use of the sequence-to-sequence paradigm. The research question we aim to address in this chapter is:

RQ4 Can an efficient memory network be designed using RNNs with attention mechanism only, without loosing performance?

We present the *Attentive Memory Network*, an end-to-end trainable, sequence-to-sequence memory network. It uses attention as its primary mechanism of selecting relevant memories. Rather than attending over the full input sequence at word-level, a hierarchical input encoder is used, which aggregates the word-level input to sentence-level representations. Subsequently, the memory module attends over sentence representations, rather than over the words, which yields a compact model.

In the last research chapter, Chapter 8, byte-level models are considered for solving machine reading tasks in morphologically rich languages — such as Turkish and Russian. In these languages, many more unique words exist than in English due to highly productive inflection, and prefix and suffix mechanisms. This may set back word-level models, since vocabulary sizes too big to allow for efficient computing may have to be employed. Multiple alternative input granularities have been proposed to avoid large input vocabularies, such as morphemes, character n-grams, and bytes. Of these, bytes are especially advantageous as they provide a universal encoding format across languages, and allow for a small vocabulary size, which, moreover, is identical for every input language, making byte-level input especially suitable for comparing models across languages. Therefore, in this chapter, we investigate the following research question:

RQ5 Is it advantageous, when processing morphologically rich languages, to use bytes rather than words as input and output in a machine reading task?

1.2 Main contributions

To sum up, the main contributions of these thesis are as follows.

Part I

Chapter 4 Ad Hoc Monitoring of Vocabulary Shifts over Time

Task We introduce the task of ad hoc monitoring of vocabulary shifts over time.

Algorithm We present an algorithm for tracking shifting vocabularies over time given a small set of seed words, and systematically evaluate the results it produces over a substantial period of time (over four decades).

Evaluation set As the task of monitoring shifting vocabularies over time for an ad hoc set of seed words is, to the best of our knowledge, a new one, we construct our own evaluation set, which we make publicly available.

Part II

Chapter 5 Short Text Similarity with Word Embeddings

Algorithm We present an algorithm of semantic matching for short texts based on existing word embeddings. We show on a publicly available evaluation set commonly used for the task of semantic similarity that our method outperforms baseline methods that work under the same conditions.

Chapter 6 Siamese CBOW

Algorithm We present *Siamese CBOW*, a novel neural network architecture for optimizing word embeddings for the task of being averaged to yield short text representations. We show the robustness of the Siamese CBOW model by evaluating it on 20 datasets stemming from a wide variety of sources.

Software The code of the Siamese CBOW method is released under an open source license.

Part III

Chapter 7 Attentive Memory Networks for Natural Language Understanding

Algorithm We present the *Attentive Memory Network*, a neural network architecture for solving machine reading tasks. We show on 20 datasets commonly used for evaluating machine reading algorithms that the Attentive Memory Network solves 18 of the tasks defined by them while using considerably fewer computations than existing memory networks.

Software The code of the Attentive Memory Networks is released under an open source license.

Chapter 8 Byte-level Machine Reading across Morphologically Varied Languages

Datasets We provide a platform for comparing machine reading models across different types of languages, by releasing 2 large machine reading datasets, one in Turkish, one in Russian — analogical to an already existing one in English. The three datasets combined provide the first data collection available, to our knowledge, for comparing machine reading algorithms on a single machine reading task, across fundamentally different languages.

Algorithm We investigate the efficacy of using bytes as input. We implement 4 byte-level models, representing the major types of machine reading models (vanilla RNN, convolutional RNN, hybrid word-byte-level, memory networks) and introduce a new seq2seq variant, called encoder-transformer-decoder. We show that, for all languages considered, there are models reading bytes outperforming the current state-of-the-art word-level baseline. Moreover, the newly introduced encoder-transformer-decoder performs best on the morphologically most involved dataset, Turkish.

1.3 Thesis overview

Before presenting the research chapters 4, 5, 6, 7, 8, as discussed above, we cover shared background in Chapter 2, as well as related work. We conclude in Chapter 9.

1.4 Origins

The research in this thesis is based on the following publications:

Chapter 4 *Ad Hoc Monitoring of Vocabulary Shifts over Time*, CIKM 2015, Kenter, Wevers, Huijnen, and de Rijke [83]

The research question of this paper was inspired by a discussion with Wevers and Huijnen, who also contributed to additional discussions about the model, and played a role in constructing the dataset. Kenter wrote and ran the code for the experiments. All authors contributed to the text, Kenter did most of the writing.

Chapter 5 *Short Text Similarity with Word Embeddings*, CIKM 2015, Kenter and de Rijke [79]

De Rijke contributed to discussions concerning every aspect of the paper. Kenter wrote and ran the code for the experiments. Both authors contributed to the text, Kenter did most of the writing.

Chapter 6 *Siamese CBOW*, ACL 2016, Kenter, Borisov, and de Rijke [84]

All authors contributed to discussion about the experiments. Borisov advised on implementation matters. Kenter wrote and ran the code for the experiments. All authors contributed to the text, Kenter did most of the writing.

Chapter 7 *Attentive Memory Networks for Natural Language Understanding*, CAIR'17, Kenter and de Rijke [80]

De Rijke contributed to discussions concerning every aspect of the paper. Kenter wrote and ran the code for the experiments. Both authors contributed to the text, Kenter did most of the writing.

Chapter 8 *Byte-level Machine Reading across Morphologically Varied Languages* — under review — Kenter, Jones, and Hewlett [86]

The research was performed during a research internship at Google Research, in Mountain View, California. Jones was supervisor of the internship and advised on implementation matters. Kenter implemented and ran original experiments. Jones performed additional experiments. Hewlett contributed to every aspect of the process. All authors contributed to the text, Kenter did most of the writing.

Indirectly, the thesis also builds on the following joint work:

- *Neural Networks for Information Retrieval*, full day tutorial SIGIR 2017, Kenter, Borisov, Van Gysel, Dehghani, de Rijke, and Mitra [85]
- *Hierarchical Re-estimation of Topic Models for Measuring Topical Diversity*, ECIR 2017, Azarbonyad, Dehghani, Kenter, Marx, Kamps, and de Rijke [9]
- *Evaluating document filtering systems over time*, Information Processing and Management (IPM) 2015, Kenter, Balog, and de Rijke [82]
- *Design and implementation of ShiCo: Visualising shifting concepts over time*, HistoInformatics 2016, Martinez-Ortiz, Kenter, Wevers, Huijnen, Verheul, and van Eijnatten [114]

- *ShiCo: A Visualization Tool for Shifting Concepts Through Time*, DH Benelux 2016, Martinez-Ortiz, Kenter, Wevers, Huijnen, Verheul, and van Eijnatten [115]
- *Concepts Through Time: Tracing Concepts in Dutch Newspaper Discourse (1890-1990) using Word Embeddings*, DH 2015, Wevers, Kenter, and Huijnen [160]
- *Filtering Documents over Time for Evolving Topics - The University of Amsterdam at TREC 2013 KBA CCR*, TREC 2013, Kenter [78]
- *Multilingual Semantic Linking for Video Streams: Making "Ideas Worth Sharing" More Accessible*, WWW 2013 (WoLE2013), Odijk, Meij, Graus, and Kenter [123]
- *xTAS and ThemeStreams*, DIR 2013, de Rooij, Kenter, and de Rijke [36]
- *Time-Aware Chi-squared for Document Filtering over Time*, SIGIR 2013 (TAIA 2013), Kenter, Graus, Meij, and de Rijke [81]
- *Context-Based Entity Linking–University of Amsterdam at TAC 2012*, TAC 2012, Graus, Kenter, Bron, Meij, and de Rijke [53]

2 Background

In this chapter we discuss concepts commonly used throughout this thesis. In particular, we introduce word embeddings, which are used in Chapter 4, 5, and the word2vec architecture, which is varied upon in Chapter 6. Subsequently, recurrent neural networks and sequence-to-sequence models are discussed, as they are used in Chapters 7 and 8.

2.1 Word embeddings and word2vec

Word embeddings are vector representations of words that can be optimized to have properties reflecting the semantics of the words they represent as discussed below. There are many different ways of obtaining word embeddings [35, 118, 119, 124]. We detail the inner workings of one algorithm in particular, word2vec [118, 119], as the Siamese CBOW architecture presented in Chapter 6 builds on it.

2.1.1 Word embeddings

Word embeddings are vector representations of words [35]. That is, words are represented by a fixed number of floating point values – vectors. For many applications, including the ones in this thesis, it is desirable for the word vectors to have the property that semantically similar words have similar vectors. In other words, words that have approximately the same meaning should have approximately the same vectors. An alternative way of thinking about this is to regard words as being embedded in a multi-dimensional space, the number of dimensions being the fixed number of floating point values, in which semantically related words cluster together.

Figure 2.1 provides a visualization of 3-dimensional word embeddings. A vector representation of a word is denoted as $\overrightarrow{word}$. The figure illustrates that word vectors of semantically closely related words, *newspaper* and *magazine*, are closer to each other than to the vector of the unrelated word, *biking*.

As word vectors share dimensions, they can be considered to be *distributed representations* of words – as opposed to *localist representations*, where every word would have a single unique value associated with it.

To get word vectors that have the property described above, different algorithms have been proposed, such as word2vec [118, 119] and GloVe [124].

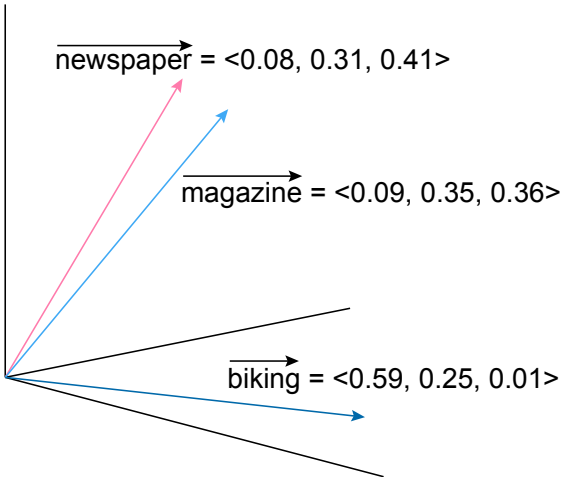


Figure 2.1: Visualization of 3-dimensional word embeddings.

In word2vec, an iterative procedure is implemented, where after initializing the word vectors, small incremental changes are applied to the vectors at every iteration, aimed at improving the desired semantic properties, until convergence.

An alternative way of getting word embeddings, called GloVe, is proposed in [124]. It is based on global matrix factorization. As such, it is close to LSA [40], but instead of a document-word co-occurrence matrix, a word-word co-occurrence matrix is used. GloVe avoids the large computational cost of, e.g., LSA by not building the full co-occurrence matrix, but training directly on the non-zero elements in it. As a cost function, the model uses a weighted least squares variant. The weighting function has two parameters, an exponent and a maximum cut-off value that influence the performance.

The vectors computed by a word embedding algorithm can be used in downstream applications. If a downstream application does not optimize the vectors any further, they are commonly referred to as *pre-trained embeddings*. In Chapter 4 we use pre-trained vectors as obtained by the word2vec algorithm. In Chapter 5 we use both pre-trained word2vec vectors and pre-trained GloVe vectors, which we refer to as *out-of-the-box* vectors.

As the Siamese CBOW architecture presented in Chapter 6 can be seen as an adaptation of word2vec, we detail this architecture in the next section.

2.1.2 Word2vec

Word2vec [118, 119] is based on the distributional hypothesis which states that words occurring in similar contexts tend to have similar meanings [59], an intuition immortalized by J.R. Firth as *"You shall know a word by the company it keeps"* [46]. In word2vec, this idea is implemented by iteratively updating vectors for words to be closer to the vectors of the words surrounding them. The algorithm needs a corpus of texts, such as web pages or newspaper articles, as input. At every step a word is sampled from the corpus. The vectors of the words surrounding it are summed and a dot product between the sum vector and the vectors of all words in the vocabulary

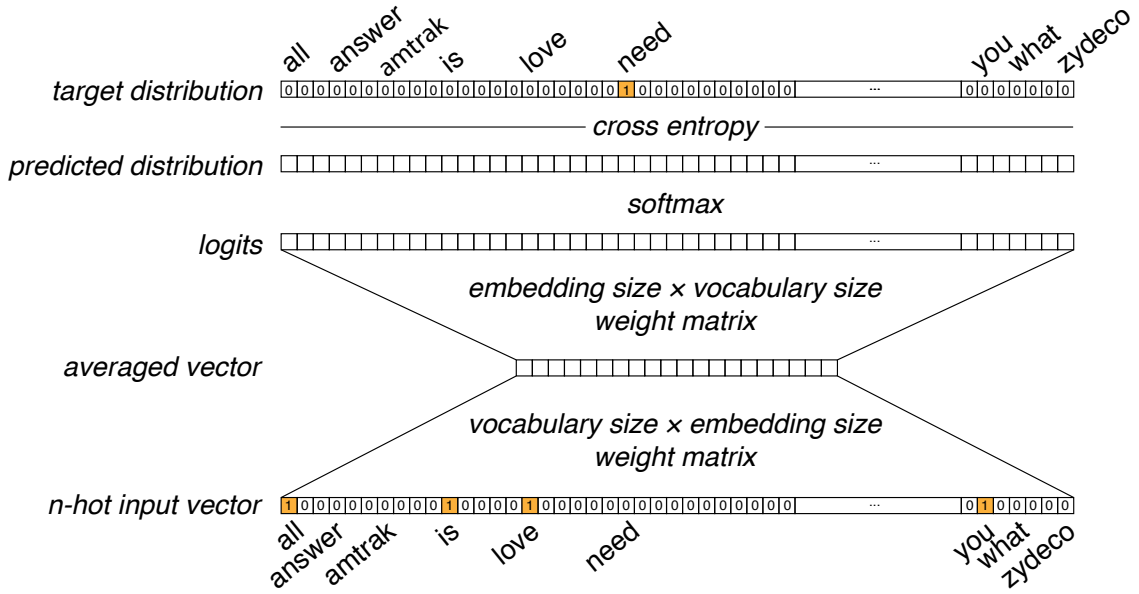


Figure 2.2: Graphical display of the word2vec CBOW architecture, where the input sentence is “all you need is love”.

is carried out. The resulting vector is normalized by applying the softmax function, that gives a vocabulary-sized vector of floating point values that sum up to one. The normalized vector is interpreted as being a probability distribution over words in the vocabulary. This distribution is intended to predict the word originally sampled, i.e., it should be close to a target distribution that is 1 at the position the sampled word has in the vocabulary, and 0 elsewhere. The error between the predicted distribution and the target distribution is expressed by a cross entropy loss function. The parameters of the model, the embedding matrices, are optimized with back propagation, aimed at minimizing the loss function.

Figure 2.2 presents a graphical representation of the word2vec Continuous Bag of Words (CBOW) architecture. It represents a neural network processing the input at the bottom to produce the output, at the top. The input to the network in this example is *“all you need is love”*. The context words, *all*, *you*, *is* and *love* are presented to the network as an n-hot input vector. This vector has as many values as there are words in the vocabulary, and every position in it corresponds to a particular word type. If a word occurs in a context, the corresponding value in the vector is set to 1, and it is 0 otherwise. This input vector is multiplied by a weight matrix that has as many rows as there are words in the vocabulary, and as many columns as the dimensionality of the embedding space (the dimensionality is a hyperparameter of the model, and is typically in the range of 200 to 500). The multiplication results in a vector that is the sum of the word vectors of the context words. The sum vector is multiplied again by a weight matrix, which projects it back again to a vocabulary size vector. The numbers in this vector are interpreted as unnormalized probabilities, also called *logits*. The softmax

function is applied to the logits. For every value $l[i]$ in logit vector l we have:

$$\text{softmax}(l[i]) = \frac{e^{l[i]}}{\sum_{j=1}^{|l|} e^{l[j]}}. \tag{2.1}$$

This results in a predicted distribution over words in the vocabulary, from which a cross entropy loss function is calculated as follows:

$$L_{\textit{cross entropy}} = -\sum_{w \in V} p(w) \log \hat{p}(w), \tag{2.2}$$

where $\hat{p}(w)$ is the predicted probability of word w in vocabulary V, and and $p(w)$ is its probability in the target distribution.

As can be seen from Figure 2.2, there are two embedding matrices in the model: both matrices with vocabulary size $\times$ embedding size elements. One is at the input side of the network, and one is at the output side. The embeddings at the output side are the ones usually used in applications.

The architecture displayed in Figure 2.2 is a schematic overview of word2vec, to illustrate the intuition of its workings. In the top layer of the network, a softmax is performed across the entire vocabulary. As the vocabulary can grow very big, in the hundreds of thousands, this operation can become too computationally expensive for practical purposes. To avoid computing a softmax over the entire vocabulary, a hierarchical softmax can be applied on a Huffman tree representation of the vocabulary, which saves calculations, at the potential loss of some accuracy. An alternative strategy to get better embeddings is negative sampling, where, instead of performing a softmax over the entire vocabulary, a softmax is calculated over positive examples (context words) and negative examples (words sampled from the corpus). The negative sampling approach is used in the Siamese CBOW architecture, proposed in Chapter 6.

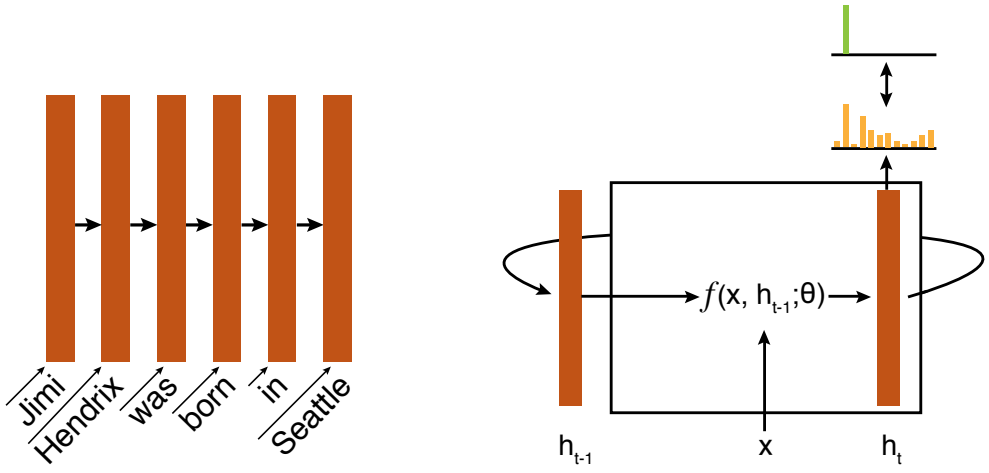
There is an alternative architecture to the word2vec CBOW architecture described above, which is called Skip-gram. It works similar to CBOW, the difference being that the inputs and outputs are swapped. The sampled word is presented as input to the network (*need* in the example above), and the network has to predict the context words (*all*, *you*, *is* and *love*). In our work in Chapter 5 we use word2vec CBOW and Skip-gram, as well as GloVe word embeddings.

2.2 Recurrent neural networks and sequence-to-sequence models

Recurrent neural networks (RNNs), in particular sequence-to-sequence models [145], play an important role in Chapters 7 and 8. In this section we discuss their general workings and the related concept of *attention*.

2.2.1 Recurrent neural networks

Figure 2.3a shows a graphical representation of a RNN. The vertical bars represent the internal states of a recurrent cell. The input sentence is presented as a sequence of word



(a) Recurrent neural network.

(b) Recurrent cell.

Figure 2.3: Recurrent neural network an recurrent cell.

vectors to the model. The input vectors are read sequentially, and at every step, the network computes a function that has two inputs, a word vector and a state vector, and one output, a new state vector. Recurrent neural networks (RNNs) are called *recurrent* because at step t, the state vector the network computed at step $t-1$ is used to compute a new state vector from.

Figure 2.3b zooms in on a recurrent cell. The input vector is represented by x. The recurrent cell computes a new state h_t as a function of an input representation x and an internal state vector, also called its *hidden state*, h_{t-1} as:

$$\mathbf{h}_t = f(\mathbf{x}, \mathbf{h}_{t-1}; \theta), \tag{2.3}$$

based on internal parameters θ.

A common scenario for RNNs is to be trained in a language modeling setting, where at every step, the task of the network is to predict the next word, as follows:

$$\hat{w}_t = \arg\max_{\mathbf{w} \in \mathbf{V}} \frac{e^{h_t \cdot \mathbf{w}}}{\sum_{\mathbf{w}' \in \mathbf{V}} e^{h_t \cdot \mathbf{w}'}}, \tag{2.4}$$

where $\hat{w}_t$ is the predicted word at step t, calculated using the hidden state at step t, h_t, and $\mathbf{V}$, a matrix of word vectors w representing words in the output vocabulary. The network computes a probability distribution over its vocabulary, by doing a dot product between the hidden state h_t and the output embedding matrix, and applies a softmax (cf. Equation 2.1). From the distribution predicted this way (depicted in orange in Figure 2.3b) the cross entropy loss is computed by comparing it to a target distribution (displayed in green in Figure 2.3b), according to Equation 2.2.

The function f in Equation 2.3 can be implemented in many ways, for example as an Long Short-Term Memory (LSTM) [68] or Gated Recurrent Unit (GRU) cell [32]. The initial hidden state $\mathbf{h}_0$ is usually a 0-vector.

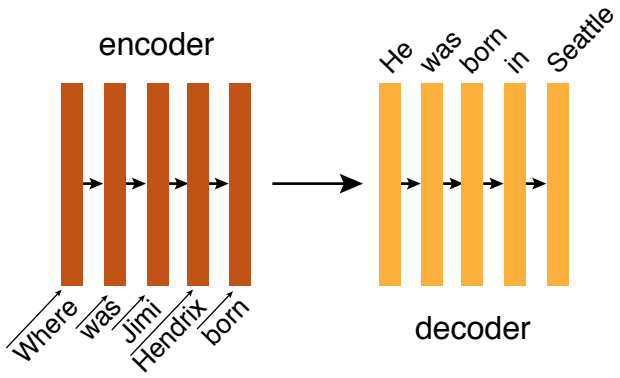


Figure 2.4: Sequence-to-sequence model

2.2.2 Sequence-to-sequence models

A sequence-to-sequence model is a combination of two RNNs. The first RNN, called the encoder, reads an input sequence, while a second RNN generates an output sequence. The architecture was originally conceived of in a machine translation setting [145] where the input is a sentence in a source language, and the desired output is a translation of the input sentence in a target language. It has, however, been applied to many settings that have a sequence of symbols as input and another sequence of symbols as output, such as question answering [64, 164], query suggestion [142] and dialogue systems [137, 149]

Figure 2.4 presents a graphical display of a sequence-to-sequence model. The encoder repeatedly applies Equation 2.3, based on its parameters $\theta_{encoder}$, to the input word embeddings. The embeddings are presented to the network as a matrix, $\mathbf{X}^{enc} = [\mathbf{x}_1^{enc}, \mathbf{x}_2^{enc}, \ldots, \mathbf{x}_n^{enc}]$, where every $\mathbf{x}_i$ is a word vector. The encoder yields an $n \times d^{enc}$ matrix $\mathbf{H}^{enc} = [\mathbf{h}_1^{enc}, \mathbf{h}_2^{enc}, \ldots, \mathbf{h}_n^{enc}]$ of n hidden states of dimension d^{enc}, depicted in brown in Figure 2.4. A second RNN, called the decoder, generates output according to Equation 2.3, where the initial hidden state, instead of a vector of zeros, is the last hidden state of the encoder $\mathbf{h}_n^{enc}$. The hidden states of the decoder are depicted in orange in Figure 2.4.

In a sequence-to-sequence model, the hidden states of the encoder are usually not used to predict the next word, or in computing a loss function. Rather, when the encoder has read the input, the decoder generates output at every step t, typically using its hidden state, h_t^{dec}, by calculating a softmax over the vocabulary, following Equation 2.4.

At training time, the embedding of the correct word — the word that should have been returned — is usually given as input to the recurrent cell at time step $t + 1$. At test time, the embedding of the predicted word is used.

2.2.3 Attention

An attention mechanism was proposed in [10], which gives the decoder access to the hidden states of an encoder. Instead of using Equation 2.3 to produce a new hidden state dependent only on the input, the computation now also depends on $\mathbf{H}^{att}$, the states to

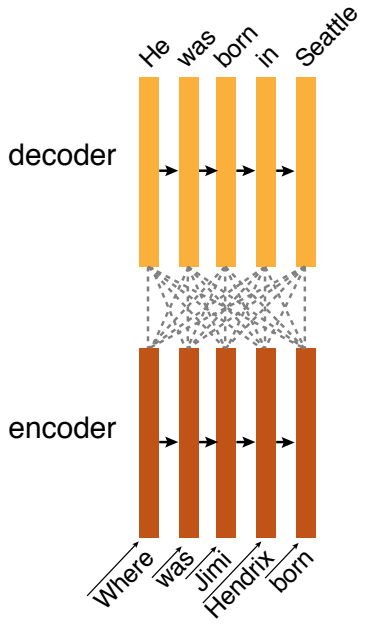


Figure 2.5: Sequence-to-sequence model with attention mechanism.

attend over, typically the states of the encoder. Following, e.g., [109, 150], we have:

$$\begin{aligned} \mathbf{h}_t^{dec} &= g(\mathbf{x}^{dec}, \mathbf{H}^{att}, \mathbf{h}_{t-1}^{dec}) \\ &= \mathbf{W}_{proj} \cdot \mathbf{d}_t || \hat{\mathbf{h}}_t^{dec}, \end{aligned} \tag{2.5}$$

where $||$ is the concatenation operator, $\hat{\mathbf{h}}_t^{dec} = f(\mathbf{x}^{dec}, \mathbf{h}_{t-1}^{dec}; \theta^{dec})$ from Equation 2.3 and $\mathbf{d}_t$ is calculated from $\mathbf{H}^{att}$ by:

$$\begin{aligned} \mathbf{d}_t &= \sum_{i=1}^{n} a_{t,i} \mathbf{h}_i^{att} \\ \mathbf{a}_t &= \text{softmax}(\mathbf{u}_t) \\ \mathbf{u}_{t,i} &= v^T \tanh(\mathbf{W}_1 \mathbf{h}_i^{att} + \mathbf{W}_2 \mathbf{h}_t^{dec}), \end{aligned}$$

where $\mathbf{h}_i^{att}$ is the i-th state in $\mathbf{H}^{att}$, and the vector v and matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ are extra parameters learned during training. From the hidden state produced this way, output can be generated by applying Equation 2.4 as usual.

Figure 2.5 depicts a sequence-to-sequence model with attention, where the attention is represented by dashed gray lines. The figure illustrates that the decoder, at every decoding step, has access to all the hidden states of the encoder.

2.3 Outlook

The methods discussed in this chapter are used throughout the rest of the thesis. In Chapter 4 word2vec word embeddings are trained on Dutch newspaper articles, and employed by an algorithm that aims to capture changes in word usage over time.

The question underlying the research in Chapter 5 is how pre-trained word vectors can be used to calculate similarity between short texts. A semantic similarity classifier is trained on features derived from word2vec and GloVe vectors that were published with previous research. In addition, vectors trained by ourself on Wikipedia data are employed, to test the sensitivity of both methods to different parameter settings.

The Siamese CBOW network presented in Chapter 6 builds on the word2vec CBOW architecture. Where word2vec aims to minimize the distance between vectors of words and the vectors of the words surrounding them in context, we aim to do the same on sentence level. I.e., we optimize sentence vectors to be near the vectors representing sentences around them in the training corpus.

RNNs and sequence-to-sequence models play a central role in machine reading research, the topic of the last two research chapters of this thesis. We propose a novel network architecture based on sequence-to-sequence models and the attention mechanism described above in Chapter 7. In the final research chapter, Chapter 8, multiple machine reading models, applied to byte-level input, are compared, and a novel adaption of a sequence-to-sequence model with attention, called encoder-transformer-decoder, is proposed.

3 Related work

In this chapter, we present work related to the research presented in this thesis. We discuss the related work per chapter.

3.1 Words

In this section we discuss work related to the first part of this thesis that deals with text understanding at the level of words.

3.1.1 Ad hoc monitoring of vocabulary shifts over time

In Chapter 4 we describe a method for tracking changes in word usage over time. While this task, to the best of our knowledge, has not been studied before, our work touches on multiple branches of existing research. We describe work related to each branch in turn below.

Change in vocabulary over time with topic models Topic models, like LDA and PLSA have been used extensively to monitor topics over time, starting with seminal work in [20, 155]. In [57] topic models are used to model the history of scientific ideas through time. The setting is similar, but different to the one we address, as word distributions of topics are inferred from the entire dataset and vocabulary shift is not modeled directly. Rather, changes over time are modeled as shifts in the probability distribution of topics over the years. A related setting is addressed in [51] where topics and vocabulary are monitored over time. In [60] a version of LDA is presented for monitoring evolving topics in scientific literature, taking citations into account.

The most important difference between topic model-based approaches, such as the ones discussed above, and the method we present in Chapter 4 is that our approach allows for an ad hoc setting. Topic models aim to infer a fixed set of latent topics from a corpus. This is the case even when non-parametric methods are employed [22], for which the number of topics is not fixed but inferred from the data. The non-parametric models allow for more flexibility, but once the algorithm has run, there is a fixed set of topics it has inferred. The inferred topics can be investigated to see interesting patterns over time, but if the topic of interest to the user is not in the inferred set of topics, there is no way around this.

Evaluation, from the perspective of the topics, is typically extrinsic, rather than intrinsic. The top-10 words for a selection of topics is shown in [57] but not evaluated. In [51] perplexity of the inferred topics is used as evaluation metric.

History of ideas In humanities research changing vocabularies are studied as well, in particular in the field of intellectual history or the "history of ideas". In this context, a distinction is made between the *intension* of an idea and its *extension* in [18]. The intension is the meaning of a concept, the extension comprises its mentions: "The extension of [a] concept differs through time. When confronted with certain changes in extension in the data, one likely conjecture is that the meaning of the concept [...] has changed" [18]. In our work in Chapter 4 we regard the words used to denote this meaning as its extension, rather than sentences or entire articles as in, e.g., [154]. Although the intension of a concept changes as its extension changes, we assume that the intension changes gradually over time (e.g., the intension of the concept of nuclear weapons is relatively stable over time, while the names of particular instances, and the words used to refer to nuclear weapons might change over time as the techniques involved evolve). By monitoring shifts in vocabularies over time, we aim to monitor shifts in the extension of a concept. We assume that the intension of a concept is continuous enough over time to allow for such monitoring. By adhering to this assumption we follow, e.g., Kuukkanen who introduces a distinction between the core of a concept and its margins when discussing conceptual change: "the main idea is that conceptual continuity requires the stability of the core of the concept, but not necessarily that of the margin, which is something that enables a description of context-specific features" [101]. While we do not explicitly model the core or margin of concepts, we do assume conceptual continuity in our research in Chapter 4.

Topic detection and tracking The goal of topic tracking systems is to extract documents from a given stream of documents that are relevant to a set of topics of interest. Topics, in this setting, are typically events [7] or entities [47]. As events and entities may evolve over time, many adaptive document filtering algorithms have been proposed [16, 78, 132]. A sliding window approach is used on a stream of documents in [132], a component we also use in our method of monitoring shifting vocabularies over time in §4.2.

Document filtering algorithms typically contain a profile of the events or entities they monitor in the form of a (weighted) list of words which can adapt over time. Maintaining such a profile is clearly analogical to the task addressed in our work in Chapter 4, although we aim to track the words that are used to describe the meaning of a concept, rather than events or entities. Furthermore, it is important to note that in our present setting of vocabulary tracking the aim is to list terms that are semantically very similar to one another, while in the document filtering case it is beneficial for a filtering profile to cover a range of aspects as diverse as possible concerning the event or entity in question.

Change of word semantics over time Research on detecting semantic shifts for words has seen a surge of interest recently. In [89] word vectors are trained on a corpus

spanning over more than a century, with word2vec [118]. The vectors are trained on an incrementally growing time window, rather than a sliding window as we propose. Several examples are shown to illustrate that dramatic semantic changes over time can be detected by monitoring the distance between the vector of a word in the initial model that contains the least recent documents, and the vector of the word in models trained on documents published in more recent periods in time. Similarly, in [162] words are monitored over centuries. A number of examples is presented that show that changes in meaning as well as additional meanings of words can be detected.

In [56] semantic change between words is measured with a distributional semantics method. The Google Books Ngram corpus[1] is used to construct co-occurrence vectors of words in two decades (the 1960s and the 1990s, which is roughly the same time frame we use in our experiments in §4.3). The task is to detect whether or not words have undergone a drastic semantic change, and human annotators were asked to annotate for a hundred words whether or not their meaning changed over the decades. In [66, 69] co-occurrence statistics are used to find related words to a specific term, which are monitored to find the sudden shifts in meaning.

We should note that, though monitoring the shifts in meanings of words over time is related to the setting of monitoring shifting vocabulary in Chapter 4, there is a key difference between the two. To illustrate, consider the main example used in [89]: the word "gay". The meaning of this word shifted considerably over the last century. Rather than focussing on the word "gay" itself to monitor its shift in meaning, the question we ask is: what words came in its place? Apparently, the meaning of the word "gay" evolved, and it now (largely) means something else from what it used to mean. So, which terms took its place? Which terms were used in a later time frame, to denote the meaning that was previously referred to by "gay?" Our aim is to track the *concept underlying a particular set of seed words*. Crucially, in our adaptive approach for monitoring vocabulary shifts over time, we allow the original seed words to disappear completely. However, as the task in this work is related to the one addressed in, e.g., [66, 69], we construct our baseline accordingly.

Distributional semantics We use the word2vec algorithm, described in Chapter 2, to obtain word vectors in our experiments in §4.3. This approach has proven to yield high-quality word embeddings [14, 119]. The same goes for the GloVe algorithm [124]. GloVe, however, has a limitation in that it needs considerably more resources in terms of training time and memory consumption than word2vec does, which is a drawback given the large size of our corpus. It should be noted, however, that there is no theoretical restriction on the choice of distributional semantic model in the algorithms we propose in §4.2.

Methods of evaluation The evaluation used to assess the quality of the approaches discussed above is frequently based on a small number of positive examples [57, 66, 69, 89, 98, 162]. Following [56] we perform explicit intrinsic evaluation, where we ask human annotators to judge the quality of the output of our algorithms directly.

[1]The Google Books Ngram corpus is documented at `http://storage.googleapis.com/books/ngrams/books/datasetsv2.html`.

3.2 Short texts

This part of the thesis contains two chapters. In Chapter 5, we present a method for computing semantic similarity between short texts using pre-trained word embeddings. In contrast, in Chapter 6, word embeddings are trained from scratch, by an end-to-end trainable neural network that optimizes word embeddings for the task of being averaged to represent short texts.

3.2.1 Short text similarity with word embeddings

In this section we discuss work related to the different aspects of our method of determining semantic similarity between short texts from pre-trained word embeddings.

Text-level semantics without external semantic knowledge Word embeddings, as described in Chapter 2, provide a way of comparing terms semantically, by comparing their vector representations. It is not evident, however, how longer pieces of text should be represented with them. Several approaches have been proposed to go from word-level semantics to phrase-, sentence-, or even document-level semantics.

Le and Mikolov [103] propose a variation on the word2vec algorithm for calculating paragraph vectors, by adding an explicit paragraph feature to the input of the neural network. A convolutional neural network, built on top of word2vec word embeddings, is employed for modelling sentences in [70]. Other corpus-based methods have been proposed, such as [72], in which both semantic and string distance features are employed, and [138] in which a vector space model is used. All four methods, in line with the work presented here, do not rely on external sources of structured semantic knowledge, nor on natural language resources. As such, these methods are natural baselines for our experiments in §5.3. It is problematic to reproduce the work presented in [103], however, as the original source code was not released by the authors and it is not clear, algorithmically, how the second step – the inference for new, unseen texts – should be carried out. Therefore, we omit this method as a baseline.

Many methods rely on natural language resources such as parsers. Socher et al. [139] propose recursive auto-encoders for the task of semantic textual similarity. This method relies on full parse trees for every sentence it processes. Annesi et al. [8] apply a kernel method on dependency parse tree features. Another strong method is presented in [73] where features from a dependency parser are used to train a supervised method. The latter method, to our knowledge, yields the highest performance on the MSR Paraphrase Corpus [38, 126], an evaluation set commonly used for textual similarity experiments, and the one we use in our experiments in §5.4.

Sentence representations based on word2vec word embeddings are also the focus in [88], where a convolutional neural network is trained on top of word2vec word embeddings. Convolutional networks, when applied to text, however, have proven to be successful primarily in classification tasks [88, 163] and we omit them from our experiments on semantic similarity.

Text-level semantics with external knowledge A large body of research has been directed at using sources of structured semantic knowledge like Wikipedia and WordNet

for semantic text similarity tasks. In [44, 45, 108] similar methods are proposed, using pairings of words and Wordnet-based measures for semantic similarity. Our method for aligning words as described in §5.2 draws on this work. The key difference between these approaches and ours, apart from the fact that WordNet is used, is that parsing/POS tagging is carried out [45, 108], as the WordNet-based measures are limited to comparing words having the same POS tag. Furthermore, no full-scale machine learning step is involved. All methods present one overall score, based on a threshold which is calculated through a simple regression step [45, 108] or set manually [44].

Corpus methods are combined with WordNet-based measures in [106, 117]. In [117] an IDF-weighted alignment approach, based both on WordNet-based and corpus-based similarities, is proposed. Texts are parsed and only similarities within identical part-of-speech categories are considered. Finally, a single score is calculated as an average over the maximum similarities. In [106] a WordNet similarity measure is combined with word order scores. Neither approach involves a machine learning step.

SemEval STS The SemEval 2012 Semantic Text Similarity (STS) task [2] and SemEval 2013 STS task [3] (part of *SEM'13) evaluation campaigns provide a platform for competing teams to evaluate algorithms for determining semantic text similarity. A full description of the work of all participating teams (over 30 in both years) is beyond the scope of this section. We discuss the approaches of the best-scoring teams.

The best-scoring teams in 2012 both calculate a large number of features based on a wide variety of methods. Additionally, handcrafted rules are applied that deal with currency values, negation, compounds, number overlap [134] and with literal matching [13]. The main difference with our approach, apart from the handcrafted rules, is in the features extracted, and in particular the number of additional resources required (WordNet, a dependency parser, NER tools, lemmatizer, POS tagger, stop word list [134], and WordNet, Wikipedia, Wiktionary, POS tagger, SMT system for three language pairs [13]).

In 2013, we see similar approaches where the best teams extract features from sentence pairs and use regression models to predict a similarity score. The features in [58] are based on LSA, WordNet and additional lists of related words and stopwords. In [113] features are calculated from aggregated similarity measures based on named entity recognition with WordNet and Levenshtein distance, higher order word co-occurrence similarity, the RelEx system, dependency trees and reused features of SemEval 2012 participants. Additionally, handcrafted features like lists of aliases (e.g., *USA* and *United States*) are used. A parallel between our work and both these approaches is the use of word alignment.

Finally, in [12] a method similar to the one we propose in Chapter 5 is presented, for a related, but different task of detecting semantic similarity between texts of different lengths. Next to WordNet-based features, a word alignment method is used based on word embeddings, analogical to what we propose. A crucial difference with our approach is that only a single feature is derived from this score, rather than several bins. Moreover, only a single set of word embeddings is used, while we show in our experiments in §5.4 that it is beneficial to use multiple sets.

Meta-level features As described below in §5.2.1 and §5.2.2 below, we use bin-based features to capture the characteristics of the differences between vectors and the distribution of word embeddings. This is similar to, e.g., [28] where meta-level features are proposed in a text classification setting using the kNN algorithm, to exploit the distribution of the nearest neighbor similarities and within-class cohesion.

3.2.2 Siamese CBOW: optimizing word embeddings for sentence representations

The Siamese CBOW model optimizes word embeddings for the task of being averaged to represent short texts, by training them from scratch in an end-to-end trainable neural network. A distinction can be made between supervised approaches for obtaining representations of short texts, where a model is optimised for a specific scenario, given a labeled training set, and unsupervised methods, trained on unlabeled data, that aim to capture short text semantics that are robust across tasks. In the first setting, word vectors are typically used as features or network initializations [70, 79, 136, 168]. Our work can be classified in the latter category of unsupervised approaches.

Many models related to the one we present here are used in a multilingual setting [62, 63, 102]. The key difference between this work and ours is that in a multilingual setting the goal is to predict, from a distributed representation of an input sentence, the same sentence in a different language, whereas our goal is to predict surrounding sentences.

Wieting et al. [161] apply a model similar to ours in a related but different setting where explicit semantic knowledge is leveraged. As in our setting, word embeddings are trained by averaging them. However, unlike in our proposal, a margin-based loss function is used, which involves a parameter that has to be tuned. Furthermore, to select negative examples, at every training step, a computationally expensive comparison is made between all sentences in the training batch. The most crucial difference is that a large set of phrase pairs explicitly marked for semantic similarity has to be available as training material. Obtaining such high-quality training material is non-trivial, expensive and limits an approach to settings for which such material is available. In our work, we leverage unlabeled training data, of which there is a virtually unlimited amount.

As detailed in §6.2, our network predicts a sentence from its neighbouring sentences. The notion of learning from context sentences is also applied in [92], where a recurrent neural network is employed. Our way of averaging the vectors of words contained in a sentence is more similar to the CBOW architecture of word2vec [118], in which all context word vectors are aggregated to predict the one omitted word. A crucial difference between our approach and the word2vec CBOW approach is that we compare sentence representations directly, rather than comparing a (partial) sentence representation to a word representation. Given the correspondence between word2vec's CBOW model and ours, we included it as a baseline in our experiments in §6.3. As the Skip-gram architecture has proven to be a strong baseline too in many settings, we include it as well.

Yih et al. [167] also propose a siamese architecture. Short texts are represented by tf-idf vectors and a linear combination of input weights is learnt by a two-layer fully connected network, which is used to represent the input text. The cosine similarity

between pairs of representations is computed, but unlike our proposal, the differences between similarities of a positive and negative sentence pair are combined in a logistic loss function.

Finally, independently from our work, Hill et al. [67] also present a log-linear model. Rather than comparing sentence representations to each other, as we propose, words in one sentence are compared to the representation of another sentence. As both input and output vectors are learnt, while we tie the parameters across the entire model, Hill et al. [67]'s model has twice as many parameters as ours. Most importantly, however, the cost function used in [67] is crucially different from ours. As words in surrounding sentences are being compared to a sentence representation, the final layer of their network produces a softmax over the entire vocabulary. This is fundamentally different from the final softmax over cosines between sentence representations that we propose. Furthermore, the softmax over the vocabulary is, obviously, of vocabulary size, and hence grows when bigger vocabularies are used, causing additional computational cost. In our case, the size of the softmax is the number of positive plus negative examples (see §6.2.1). When the vocabulary grows, this size is unaffected.

3.3 Documents

We present two methods for machine reading at a document level. In the machine reading task, a program is presented with a text and has to answer questions about it without referring to external sources of knowledge. Machine reading is a much-studied domain [31, 64, 65]. It is related to question answering, the difference being that in question answering, external domain or world knowledge is typically needed to answer questions [42, 120, 165], while in machine reading answers should be inferred from a given text. In Chapter 7 an Attentive Memory Network (AMN) is proposed to address the machine reading task in an efficient way. Chapter 8 deals with languages other than English, in particular morphologically richer languages. Multiple machine reading architectures are compared, both on an already existing English dataset and on two newly presented datasets in Turkish and Russian.

3.3.1 Attentive memory networks for natural language understanding

The AMN proposed in Chapter 7 is a sequence-to-sequence model with a hierarchical input encoder and an additional memory module. Hierarchical encoders are employed in a dialogue setting in [135] and for query suggestion in [142]. In both works, the hierarchical encoder is also trained, for every input sentence, to predict every next input sentence, a setting we did not experiment with.

We build on previous work on memory networks [144, 147, 158], in particular on dynamic memory networks [99, 164]. Memory networks are an extension of the standard sequence-to-sequence architecture as described in Chapter 2. Their distinguishing feature is a memory module added between the encoder and decoder. As they are typically applied in question answering settings, there are two encoders, one for a question and one for a document the question is about. The decoder does not have

access to the input but only to the memory module, which distills relevant information from the input, conditioned on the question. The key difference between the Attentive Memory Network we propose and the work in [99, 164] is in the defining component, the memory module. In [99, 164], to obtain every next memory, a GRU cell iterates over the input sequence. This leads to a memory intensive and computationally expensive architecture, since multiple cells are repeatedly being unrolled over the input sequence. The number of steps an RNN is unrolled for, i.e., the number of input representations it reads, together with the hidden state size, is the main determining factor regarding computational complexity. Therefore, we propose to obtain memories by an RNN that, rather than iterating over the entire input, only applies attention over it, which is a much cheaper operation (see §7.2).

In [144] an attention-based memory network is presented, where the input is represented as a sequence of embeddings on which attention is computed (i.e., there is no input reader). Our Attentive Memory Network differs from this work in that we do use an input reader, a hierarchical RNN. As a consequence, our memory module has fewer hidden states to attend over, which makes it more efficient. At the output side, we use GRUs to decode answers, which is different from the softmax over a dot product between the sum of attention-weighted input and question employed in [144].

3.3.2 Byte-level machine reading across morphologically varied languages

Byte- and character-level models have been applied in different settings such as text classification [170], NER and POS tagging [50], and language modeling [90], also on morphologically rich languages [33, 107].

A word-based variant of the convolutional-recurrent model described in §8.3 is proposed in [76]. The key difference is that the receptive window of the convolutions in the model we use ranges over the entire input sequence, and hence can cross word boundaries, while in the model of [76], the convolutions can only see single words. Preliminary experiments showed worse performance for the word-level convolution model, and hence we left it out of our main experiments.

The memory network we implemented is based on the work in [80, 144, 158, 166]. The output module of the model described in [158] selects memories — stored input representations — conditioned on its current input and the previously retrieved memories. However, in [144, 158] an embedding approach is used to represent the input and generate output, while in our setting RNNs are used at both stages for better comparison to the other models. In [80], a hierarchical input reader is proposed, which reads words into sentences, and transforms sentence embeddings to memory, a setting we don't explore in the experiments in §8.4. The memory network we employ in Chapter 8 is related to the reader network described in [64] and much like the encoder-reviewer-decoder model in [166], where a reviewer module is applied between encoding and decoding. A difference is that our models repeatedly attend over the document conditioned on the question, while in [64] attention is performed only once.

The encoder-transformer-decoder model presented in §8.3 is also related to the encoder-reviewer-decoder network in [166]. There are multiple differences between the two models. The encoder-transformer-decoder model has a separate question encoder,

which is absent from the encoder-reviewer-decoder model. More importantly, however, the decoder of the encoder-transformer-decoder model attends over the document encoder states, rather than over the reviewer states. Lastly, in [166] experiments are run with discriminative supervision for the reviewer model at training time, a setting we do not use on our experiments in §8.4.

In this chapter we covered the related work for the chapters to come. We now turn to the research part of this thesis, and start with the first of the three levels of understanding test, viz. the word level.

Part I

Words

4

Ad Hoc Monitoring of Vocabulary Shifts over Time

4.1 Introduction

Word meanings change over time [56, 154]. Detecting shifts in meaning for particular words has been the focus of much research recently [56, 66, 69, 89, 98, 162]. In this chapter we address the complementary problem of monitoring shifts in vocabulary over time. Rather than taking a word as an anchor to monitor its (shifts in) meaning over time, we take the meaning as an anchor, and monitor the evolving set of words that are used to denote it. As an example, consider music storage media. Nowadays, we carry music with us on iPods and mp3 players. Before that there were compact discs. Prior to cds there were records, and music cassettes. Few of the words that were used in, say, the 1950s to describe the media used for storing music are still in use today. Following this example, the question we set ourselves to answer is "what words were used previously, where nowadays the words 'mp3 player' and 'iPod' are used?" An algorithm for monitoring vocabulary shifts over time has the words "iPod" and "mp3 player" as its input, which we refer to as *seed terms*. As output it produces ranked lists of words per time period, e.g., every 5 years, of the words in that period that represent the concept underlying the initial input words. In what follows, we refer to such ranked lists of words per time period as *vocabularies*. Specifically, in this chapter, we want to answer the following research question:

RQ1 Given a corpus of time-stamped documents, a point in time and a small set of seed terms used to denote a concept in that corpus at the time specified, can we infer from the corpus which terms are used in adjacent periods in time to denote the same underlying concept?

Not all concepts evolve as dramatically as the music storage media in the example above, where the entire vocabulary changes in the course of a few decades. Often, many terms in the vocabulary remain relevant over time. A successful system for monitoring vocabulary shifts over time should strike a balance between an adaptive strategy that responds to changes in vocabulary, and a more conservative approach that keeps the vocabulary stable.

The problem setting we address is inspired by collaborations with digital humanities scholars in the field of history. Changes in discourse over time are a popular topic of studies in the humanities [55, 57, 71, 122]. Lists of keywords are usually maintained manually. However, "[f]inding the right keywords demands expert knowledge of the field of study and a great deal of perseverance and creativity" [71]. The methods for finding shifts in vocabulary over time that we propose in this chapter are aimed at automating this task in a time-aware fashion. The resulting vocabularies are returned to the humanities scholars, as an indication of changes in discourse in the underlying corpus. They may be used for exploratory ends, to discover unfamiliar relevant historical terms. Additionally, as discussed in our future work section §9.2, if the vocabularies are of sufficient quality, they can be used for time-aware query expansion in an document retrieval setting for an historical corpus.

There has been extensive work on the related but different problem of concept drift in the context of ontologies and taxonomies; see, e.g., [154]. Any semantic ontology of terms should adapt over time in order to keep up with changes in meaning of the terms it contains. In this chapter, however, we approach concept drift from a user perspective and not from an ontology perspective. This means that we do not assume pre-defined ontologies to be available and we do not aim to infer ontologies. Our primary motivation is to track evolving vocabularies over time for a user-provided topic of interest. This motivation leads to the following set of requirements:

1. **Words as retrieval unit** – Rather than outputting documents, as in a classic information retrieval scenario, an algorithm for monitoring shifts in vocabulary over time should, given a set of seed terms and a corpus, outputs words for a sequence of periods in time.
2. **Ad hoc** – An algorithm for monitoring shifts in vocabulary over time should work ad hoc. I.e., it should not be dependent on a predefined ontology or a fixed set of topics. The user should be able to provide ad hoc input at runtime. This requirement entails that very limited input of the user should be enough. Typically, one or two initial terms should suffice as an initial seed set.
3. **Broad time coverage** – An algorithm for monitoring shifts in vocabulary over time should be able to cover a substantial amount of time, at least multiple decades, long enough for interesting changes in discourse to occur.
4. **Comprehensible outcome** – The output produced by an algorithm for monitoring shifts in vocabulary over time should be easy to consume by humans. This means that the vocabularies that an algorithm yields should be limited in size, typically consisting of only a few words.

We note that an additional implication of the ad hoc requirement (requirement 2 in the list) is that no in-depth historical or domain knowledge of a user should be necessary. I.e., a user should not be required to have extensive knowledge of the concepts the input words are about nor of the underlying corpus. Rather, an effective method for monitoring shifts in vocabulary over time should provide new insights about the concepts and the corpus.

The comprehensible outcome requirement (requirement 4) entails that an optimal rate should be found for emitting vocabularies, regardless of a method's internals. If the

rate is too low, too many vocabularies are produced, which leads to too much repetition. A rate that is too high would cause interesting shifts to go unnoticed. Precursory discussions with domain experts in the area of the history of ideas revealed that five year periods were deemed optimal.[1]

We propose an algorithm for monitoring shifts in vocabularies over time given a small user-provided set of seed terms and a period of reference. As discussed in §3.1.1, this task is related to, but different from, tracking topics over time [51, 155], where topic models such as LDA and PLSA are used to monitor changes in a predefined number of topics. A crucial difference between topic modeling approaches and the method we propose is that, rather than relying on a pre-defined number of fixed topics, we allow for ad hoc queries.

Briefly, our proposed algorithm proceeds as follows. We first use distributional semantic models to infer a series of semantic spaces over time from a large body of time-stamped textual documents (cf. §2.1). We then construct semantic networks of terms based on their representation in the semantic spaces and use graph-based measures to calculate saliency of terms. Finally, we output shifting vocabularies over time — i.e., for a small set of seed words we output ranked lists of terms for a consecutive series of periods in time. The words in the vocabularies are meant to denote the same concept as the seed words do. As there is, to the best of our knowledge, no evaluation set available that allows for the intrinsic evaluation of monitoring shifts in vocabularies over time, we construct our own.

In the next section we describe our method of tracking vocabularies over time. Our experimental setup is detailed in §4.3 while the results of the experiments are presented and analyzed in §4.4.

4.2 Monitoring shifting vocabularies through time

In this section we describe our algorithm for monitoring shifts in vocabulary over time. By *vocabulary* we mean a ranked list of unique terms.

4.2.1 Overview

Our algorithms for monitoring shifts in vocabulary over time use three components: *sliding time windows*, *generation algorithm* and *aggregation algorithm*.

We use time windows of multiple years in length (we experiment with 5 and 10 year time windows in our experiments in §4.4) and extract documents from our corpus that were published within the time window. The window length is in years and every next window starts one year later than the previous window. If we use, e.g., ten-year windows, and the overall time period starts in 1950, we have a 1950–1959 window, a 1951–1960 window, etc. From the documents published within a time window we compute a semantic model using word2vec (see §2.1.2). We have one semantic model for each

[1] We note that alternatively, the optimal rate of emitting vocabularies could be determined programmatically. In theory, it could even differ between sets of seed words. The evaluation of such an approach would require extra, non-trivial annotator effort and we consider it outside the scope of the present research.

sliding window in time. The computation of the semantic models is a pre-processing step. It is done only once for a given corpus.

As mentioned in §4.1 when discussing requirement 4, the optimal period for outputting vocabularies was found to be five years. However, the sliding windows are one year apart. To get from semantic models based on time windows one year apart, to output vocabularies spanning 5 years, we use a two-stage approach. A first algorithm, which we refer to as the *generation algorithm*, outputs a series of vocabularies, one for each sliding time window, using a semantic network it maintains from the semantic models constructed from the documents in every time window. A second algorithm, which we refer to as the *aggregation algorithm*, aggregates over the vocabularies generated by the generation algorithm to produce the final vocabularies for the desired time period.

The generation algorithm uses graph-based measures to extract the most salient words from a semantic network for a given time window. The salient words are used as input to the next iteration of the algorithm. In short, the generation algorithm takes the original user-provided words as its input and *adaptively* updates this seed set by iterating over the sliding time windows.

Our algorithms for generating shifting vocabularies over time are completely unsupervised. No labelled training data is needed, and no pre-defined ontologies are required. Only a large amount of unlabelled data has to be available to derive word vectors from.

In what follows we describe three methods of generating shifting vocabularies over time. The *adaptive* method uses both the generation algorithm and the aggregation algorithm. The *non-adaptive* method uses only the aggregation algorithm to aggregate over vocabularies generated from the sliding time windows. The *hybrid* method combines the vocabularies produced by the adaptive and non/adaptive methods. As the sliding time windows are used by all three methods, we first turn to discussing these.

Sliding time windows

As detailed in §3.1.1 the intuition underlying our model for monitoring shifts in vocabulary over time is that word meanings, and the semantic relations between words, shift gradually and continuously over time [18, 101, 154]. To make use of this continuum when constructing semantic models, we divide the time period we are monitoring into multiple time windows, and calculate a semantic model from each of these windows. I.e., we extract all documents from the corpus that were published in the desired time window and train a word2vec model on their contents.

To be sensitive to rapid changes, it would be beneficial to have short time windows. However, previous research has proven that the quality of the semantic models inferred by word2vec is higher when more training data is used [118]. We solve this conflict in requirements on the size of the training data for the semantic models by using *overlapping* time windows. By taking an extended period of time, we obtain a sufficient amount of data for constructing high-quality semantic models. As the windows are only one year apart from each other, changes in the semantic relations between words can be detected between subsequent models, while the vast majority of relations will remain stable, due to the overlap.

4.2.2 Adaptive method for generating shifting vocabularies over time

In this section we describe the generation algorithm and the aggregation algorithm, for our adaptive method of monitoring shifting vocabularies over time.

Generation algorithm; generating shifting vocabularies over time

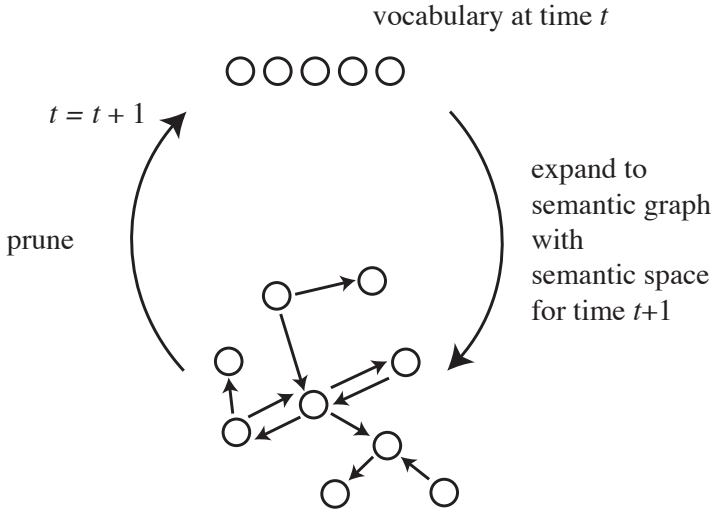


Figure 4.1: Schematic representation of the generation algorithm for generating vocabularies over time. Circles denote words.

In Figure 4.1 a schematic overview is given of the generation algorithm for generating vocabularies over time from sliding windows. The circles denote words. Every iteration consists of an expansion step and a pruning step. In the expansion step, a semantic graph is constructed from a list of seed terms and a semantic space. The semantic graph is displayed at the bottom of Figure 4.1. In the pruning step, the top terms, according to a graph-based measure, are extracted from the graph. This vocabulary, displayed at the top of Figure 4.1, is the input to the next iteration. As can be seen from this schematic overview, the original input words do not necessarily end up in the vocabulary one (or more) iterations later.

In Algorithm 1 the pseudocode for the generation algorithm is provided. At the very first iteration, the input consists of the seed set as provided by the user (Algorithm 1, line 1). As a key requirement of our method is limited effort by the user, we use only a few terms (typically one or two) as input. The outer loop is carried out K times (line 2), once for every semantic model, derived from the K sliding time windows. In the expansion step (line 4–8), we construct a weighted, directed, partial semantic graph from the set of seed terms, given the semantic space from the next time window. To do this, we obtain the related terms for every word in the seed set, with a minimum similarity ς (line 5). Per seed set term we take at most n related terms. The terms obtained in this manner are the vertices of our graph. From these vertices we construct a semantic graph (line 9). The edges in the graph are directed and weighted. We draw an edge from vertex w_i to w_j if w_j is in the list of related words of w_i. The weight

Required: $W = [w_1 \dots w_{|W|}]$: a set of seed terms
Required: Series of semantic spaces $S = [sem_1 ... sem_K]$, ordered by time
Required: ς: minimum similarity
Required: n: maximum number of terms to return
Result: List of vocabularies $v_1 \dots v_{|K|}$

```
1  v_0 = W;
2  for k ← 1 ... |K| do
3  |   vertices = [];
   |   /* expand                                                    */
4  |   foreach w ∈ v_{k-1} do
5  |   |   foreach w_related ∈ related_words(w, sem_k, ς, n) do
6  |   |   |   vertices = vertices ∪ w_related;
7  |   |   end
8  |   end
9  |   semanticNetwork = drawEdges(vertices);
   |   /* prune                                                     */
10 |   v_k = top-n nodes from semanticNetwork w.r.t. degree centrality
11 end
```

Algorithm 1: Generation algorithm: adaptively generating vocabularies from sliding time windows

on the edge is determined by the strength of the association between w_i and w_j in the semantic space. The network is partial as we do not construct an extensive network of all possible vertices (i.e., all word types in the corpus), but rather extract the part of the network in the vicinity of the seed terms. In the pruning step the top-n terms are selected relative to their degree centrality in the semantic network (line 10).

We use elementary variants of degree centrality: in-degree and out-degree. More involved measures like PageRank [26] can be considered as well, especially when larger parts of the graph are extracted, e.g., by finding related words of related words, and so on. However, preliminary experiments showed that the relation between the original seed terms and related terms of related terms can quickly become arbitrary. A method relying on such terms would run a considerable risk of topic drift.

We compute four measures of degree centrality: number of inlinks, weighted sum of inlinks, number of outlinks and weighted sum of outlinks. The choice of degree centrality measure is a parameter of our model. We discuss the effect of this parameter on the results of our experiments in §4.4.2.

Direction in time Above, we describe a forward pass, where we start with the oldest time window and progress towards the future. The same method can be applied the other way around, as would, e.g., be appropriate for the iPod example in §4.1. In Algorithm 1 this means that we start with $v_{|K|}$ in line 1, range over $k \leftarrow |K| \dots 1$ in line 2 and iterate over $w \in v_{k+1}$ in line 4.

Required: List of vocabularies $V = v_1 \dots v_{|K|}$
Required: List of time frames $T = [\tau_1 \dots \tau_{|T|}]$ for which to output vocabularies
Required: n: maximum number of terms to return
Result: List of vocabularies $v_{\tau_1} \dots v_{|T|}$

```
1 for t ← 1 ... |T| do
2 |   V' = [v ∈ V | v relevant to τ_t];
3 |   foreach v ∈ V' do
4 |   |   foreach w ∈ v do
5 |   |   |   score_w += f_weight(v, τ_t) * score_{w,v};
6 |   |   end
7 |   end
8 |   v_{τ_t} = top-n terms w sorted by score_w;
9 end
```

Algorithm 2: Aggregation algorithm: Aggregating vocabularies output by the generation algorithm to produce the final output vocabularies.

Aggregation algorithm: Producing the final output vocabularies

For each semantic space, generated from documents in overlapping time windows one year apart, the generation algorithm generates a vocabulary. If we monitor, e.g., a period of four decades, 40 vocabularies are generated, one for every overlapping window. The final output presented to the user, however, should be one vocabulary for every 5 year period, so 8 vocabularies, in the example case. To generate the final output vocabularies, we aggregate over the vocabularies generated by the generation algorithm.

The aggregation step producing the final vocabularies is distinct from the principal underlying method of generating vocabularies for all overlapping time windows. If the final vocabularies should be generated for periods of 4 or 6 years, rather than 5, the output of the generation algorithm could be used unaltered, and only one parameter needs to be changed in the aggregation algorithm.

Algorithm 2 lists the pseudocode of our method for aggregating over the vocabularies output by Algorithm 1 to produce the final output vocabularies. The first step in each iteration (line 2) is to select a set of vocabularies relevant to the time period at hand τ_t. We select all vocabularies constructed from time windows that have an overlap with τ_t. This step is needed as the length of the time windows is a parameter of the model and might not be the same as the length of τ_i. We can, e.g., use 10-year windows in the generation algorithm, while we output vocabularies for 5-year periods in the final step (i.e. the length of every period τ_t is 5).

In the inner loops of Algorithm 2 we iterate over the words in the selected vocabularies (line 3–7). We compute a score for all words, which consists of their score in vocabulary v (their degree centrality, see previous section) weighted by a weight function $f_{weight}(v, \tau)$ that assigns a weight to a vocabulary v for a time frame τ.

Vocabulary weighting function As described above, each vocabulary v_{τ_t} is constructed from a semantic space, derived from the texts of documents published in a time window, spanning a number of years. The time window has an overlap with time

period τ_t that we want to output a vocabulary for. Therefore, a weighting is needed which expresses how much vocabulary v should contribute to v_{τ_t}, the final vocabulary we output for τ_t.

The most straightforward way of weighting is to weight all vocabularies equally (i.e., apply no weighting at all). However, the central years in the period the vocabulary is derived from are most likely to best capture its semantics (e.g., if we look at the decade 1970–1979, the documents in the early 1970s might still have echoes of the late sixties, while in the late 1970s, the 1980s might already become apparent; the middle years will define the vocabulary most clearly). We implement this intuition by assuming that the probability of the contribution of years to a vocabulary v is given by a Gaussian distribution, where the mean of the distribution is the centre of the period, and we assume a standard deviation of 1.0. We model the distribution of the years in τ in a similar fashion, where the mean is the central year of τ. Given these two distributions we can use the Jensen-Shannon divergence as a proxy for the weight of v with respect to τ:

$$f_{JSD}(v, \tau) = JSD(\mathcal{N}(\mu_v, \sigma_v^2) \,\|\, \mathcal{N}(\mu_\tau, \sigma_\tau^2)),$$

where we have $\sigma_v^2 = \sigma_\tau^2 = 1$.

We note that simple overlap metrics, such as the Jaccard index, do not measure what we want, as the Jaccard index between two periods, where one period overlaps completely with the other, is always the same, regardless of whether the overlap is in the central region of the longer period or not.

4.2.3 Non-adaptive method for generating shifting vocabularies over time

Using the adaptive method for generating vocabularies, it is possible that none of the words in the original seed set are present after a few iterations. This is a desired effect, but it also introduces the risk of topic drift. I.e., the adaptive algorithm might focus on an aspect of meaning that was not intended by the user, which can cause the vocabularies being generated to drift in the wrong direction. To counter this effect, we also include runs in our experiments where the initial seed set is kept static. That is, we omit Algorithm 1, and instead output the n words most related to the words in the original seed set for every sliding time window. To generate the final vocabularies we do employ Algorithm 2.

We refer to this method, that follows a static seed set for generating shifting vocabularies over time, as *non-adaptive* method.

4.2.4 Hybrid runs

To combine the exploratory effect of the adaptive approach with the more conservative approach of the non-adaptive approach, we combine the runs of both methods of producing shifting vocabularies over time to produce *hybrid* runs. In particular, we replace the least central terms of the vocabularies produced by the non-adaptive method by the top i vocabulary terms produced by the adaptive method with respect to degree centrality. In §4.4 we report results for different values of i.

4.3 Experimental Setup

To measure the quality of the different methods of generating vocabularies over time we perform a systematic, intrinsic evaluation. We split the research question raised in §4.1 into:

RQ1.1 Given that we have an exploratory, adaptive approach and a conservative, non-adaptive approach for generating shifting vocabularies over time, can we combine the two in such a way that performance is gained over the components?

RQ1.2 How do the parameters of the generation algorithm and the aggregation algorithm affect performance?

The first research question, RQ1.1, concerns the balance between an exploratory response to change in vocabularies, which introduces the risk of topic drift, and a static, conservative approach, which does not allow for substantial evolution of vocabularies. In §4.4.1 we report on the results for our experiments regarding this question.

The second research question, RQ1.2, concerns our algorithms for generating vocabularies over time more specifically. As detailed in §4.2 we construct semantic networks to find salient terms in specific time periods. We are interested in evaluating whether, e.g., the weighting of edges is beneficial or not, or whether selecting nodes based on in-degree yields better results than using out-degree.

We analyze the performance regarding all parameters of our algorithms of generating shifting vocabularies over time in §4.4.2. In the remainder of this section we detail the aspects of our experimental setup.

4.3.1 Ground truth data

The natural ground truth data for our task of monitoring shifting vocabularies over time are the shifting vocabularies themselves. We make use of human annotators to obtain this ground truth data. The annotators' task is, given all unique words occurring in a corpus of timestamped documents, to indicate which words are relevant to a particular topic of interest in a certain time period. As it is not feasible for annotators to judge all word types in a corpus, we employ a pooling approach, which we detail below. Below, we also provide the characteristics of the seed terms.

Given a small number of seed terms, and a short text describing the underlying information need, the annotators were asked to judge terms per period on a 3 point scale: *irrelevant*, *related* and *perfect*. The *related* category is used for borderline cases in which a result is not completely off the mark, but is not exactly right either.

There were 6 annotators in total, all of whom are academic historians, well-acquainted with both the corpus and the evaluation time period. None of the authors of the paper this chapter is based on took part in the annotation effort.

Following, e.g., [56], we use the pairwise Pearson correlation to determine inter-annotator agreement. The Pearson correlation coefficient is 0.555 with a p-value $< 10^{-5}$. It shows that the judgements are highly correlated between annotators and that the averaged judgements can reliably be used to evaluate our experiments.

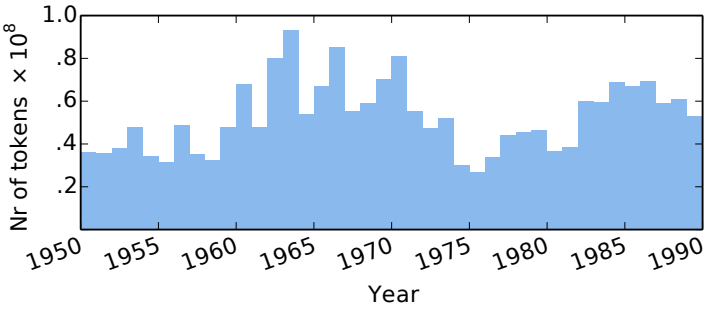


Figure 4.2: Number of tokens per year

The sets of seed terms and the ground truth annotations are publicly available. The material can be downloaded from `http://ilps.science.uva.nl/resources/shifts`.

Pooling We produce output using each of the methods for generating vocabularies over time that we consider, and all combinations of parameters. We pool these results, similar to how the runs of IR systems are pooled in a classical TREC-style evaluation [143]. In our setting, however, the unit of retrieval is a word for a given time period, rather than a document. Annotators are presented with the aggregated results of all runs combined.

Corpus Our corpus is a collection of Dutch newspapers, digitized by the Royal Library of the Netherlands.[2] We use four decades, 1950 up until 1990, as our evaluation period as this period is long enough for interesting changes to occur and modern enough for the OCR quality to be reasonable.[3]

The corpus contains 26,614,346 documents (newspaper articles) in the four decades we consider. Together, they comprise 1,940,841 unique words and 2,141,992,571 tokens. Figure 4.2 gives an overview of the numbers of tokens per year. As can be observed from this figure, the tokens are not evenly distributed across the years, but there is no bias towards either modern or historical documents.

We used the Python NLTK Punkt Sentence Tokenizer [19] and remove remaining unicode non-word characters.

Seed terms There are 21 sets of seed terms in our experiments, which are provided by Dutch historians, who are familiar with the corpus and the time period selected. The terms are inspired by their own, real-life, research questions and by observations they made from the corpus. As discussed in §4.2, an algorithm for generating vocabularies over time can run either forwards or backwards in time. It was left up to the historians to decide on the most natural direction in time, per set of seed terms. In Table 4.1 we present an overview of the seed sets, together with the direction in time. The bottom

[2]The full newspaper corpus, and more, can be queried at `http://www.delpher.nl`.

[3]No official numbers concerning the OCR quality throughout this corpus are available. Anecdotal evidence suggests though that modern material is of higher quality.

Table 4.1: Overview of the Dutch seed set words

Seed words	English explanation	Direction
cd, compactdisc	cd, compact disc	backwards
computer	computer	backwards
doping	drugs (sports-related)	backwards
efficiency, efficiëntie	efficiency	backwards
gastarbeider, gastarbeiders, immigranten	immigrants	backwards
geboortebeperking, geboorteregeling	birth control	forwards
holocaust	holocaust	backwards
internet	internet	backwards
jodenvervolging, deportatie, deportaties	persecution of Jews (in WWII)	forwards
marxisme	marxism	forwards
multinational	multinational	backwards
neger, negers, negerin, kleurling	negro, colored people	forwards
quiz	quiz	backwards
supermarkt	supermarket	backwards
waterstofbom, atoombommen, waterstofbommen, atoombom	hydrogen bomb	forwards
zelfbedieningswinkel, zelfbedieningszaak, kruidenier	self-service shop	forwards
amsterdam, rotterdam, utrecht	large Dutch cities	forwards
boek, boeken, boekje	books	forwards
koe, koeien	cows	forwards
mozart, beethoven, brahms	classical composers	forwards
viool, violen	classical instruments	forwards

5 rows in Table 4.1 list 5, so-called, a-historical seed sets. The concepts denoted by these seed sets are assumed, by the historians, to stay relatively stable over the entire evaluation period. We include the a-historical seed sets for two reasons. Firstly, we want to avoid a bias in the test set towards changing concepts, i.e., we do not want the test set to only consist of examples of which it is apparent that they evolve over time, as this would put the non-adaptive methods at an unfair disadvantage. Secondly, we want to check for over-generating, by which we mean, in this context, generating changing vocabularies while there is in fact no change. A method that is too exploratory might always find new terms and might show evolving list of words erroneously. To be able to measure such behavior, we add the a-historical seed term sets.

On average the seed term lists are 2.1 words in length. The ground truth vocabularies (i.e., the list of relevant words per time period) are 9.32 words in length on average.

4.3.2 Evaluation

Our algorithms for generating shifting vocabularies over time produce ranked lists of words. The Cranfield-style evaluation setting allows us to use traditional IR evaluation metrics suitable for evaluating ranked lists, NDCG and MAP, in addition to the standard F_1 metric.

4.3.3 Parameters and settings

We use 5-year and 10-year sliding time windows to compute semantic spaces from using the generation algorithm. Preliminary experiments showed that values between 0.6 and 0.7 are reasonable values for ς. Hence we experiment with $\varsigma \in [0.6, 0.65, 0.7]$. For degree centrality we use 4 variants, as described in §4.2.2: sum of inlinks, weighted sum of inlinks, sum of outlinks, weighted sum of outlinks.

The aggregation algorithm has only one parameter: the vocabulary weighting function. We experiment with a uniform weighting function (i.e., no weighting), and the JSD-weighting function, described in §4.2.2.

As discussed in §4.2, we use word2vec to generate word vectors for every time window. We employ default settings; Skip-gram architecture, with hierarchical softmax, vector dimensionality of 300, window size of 5, and minimum word frequency of 5.

In all experiments, the vocabulary size n is set to 10 (cf. Algorithms 1 and 2).

4.3.4 Baseline

As noted in §3.1.1, the work described in [66, 69] is related to our present setting. Following this work, we construct our baseline by using a time slice approach. However, we use neural network language models to construct semantic models to derive semantic proximity from, rather than co-occurrence measures as in [66, 69], as the computation of a full co-occurrence matrix on the corpus used in our experiments is intractable, due to its size. For every time window τ_t our baseline methods outputs the top-n most related words derived from a semantic model trained on the documents published in time window τ_t.

4.4 Results and analysis

We begin by answering our research questions and proceed by contrasting the adaptive approach and the non-adaptive approach, described in §4.2.2 and §4.2.3, respectively.

4.4.1 Hybrid vs. non-hybrid approaches

To answer RQ1.1 we conduct experiments with all methods described above and all parameter settings. Table 4.2 contains an overview of the results yielded by the best parameter setting.[4]

The key observation from Table 4.2 is that the hybrid approach outperforms both the baseline, and the adaptive method and non-adaptive method separately, on all metrics, regardless of the value of i. It is important to note that the parameter setting reported in Table 4.2 consistently yields the highest results on all metrics for the hybrid method, regardless of the value of i.

Table 4.2: Results for JSD weighting with 10-year periods, $\varsigma = 0.65$, in-degree over weighted edges. Statistically significant differences from the baseline, measured with a two-tailed paired t-test, is marked for $p < 0.02^{\dagger}$ and $p < 10^{-6\ddagger}$.

Method	F_1	p	r	NDCG	MAP
hybrid ($i = 1$)	$0.384^{\ddagger}$	$0.537^{\ddagger}$	$0.406^{\ddagger}$	$0.646^{\ddagger}$	$0.343^{\ddagger}$
hybrid ($i = 2$)	$0.391^{\ddagger}$	$0.544^{\ddagger}$	$\mathbf{0.414}^{\ddagger}$	$0.650^{\ddagger}$	$\mathbf{0.346}^{\ddagger}$
hybrid ($i = 3$)	$\mathbf{0.392}^{\ddagger}$	$\mathbf{0.548}^{\ddagger}$	$0.411^{\ddagger}$	$\mathbf{0.653}^{\ddagger}$	$0.345^{\ddagger}$
hybrid ($i = 4$)	$0.389^{\ddagger}$	$0.545^{\ddagger}$	$0.405^{\ddagger}$	$0.651^{\ddagger}$	$0.343^{\ddagger}$
hybrid ($i = 5$)	$0.385^{\ddagger}$	$0.541^{\ddagger}$	$0.399^{\ddagger}$	$0.649^{\ddagger}$	$0.339^{\ddagger}$
adaptive	0.344	$0.551^{\ddagger}$	0.298	$0.514^{\dagger}$	$0.237^{\dagger}$
non-adaptive	$0.367^{\ddagger}$	$0.521^{\ddagger}$	$0.389^{\ddagger}$	$0.630^{\ddagger}$	$0.332^{\ddagger}$
baseline	0.303	0.450	0.296	0.554	0.266

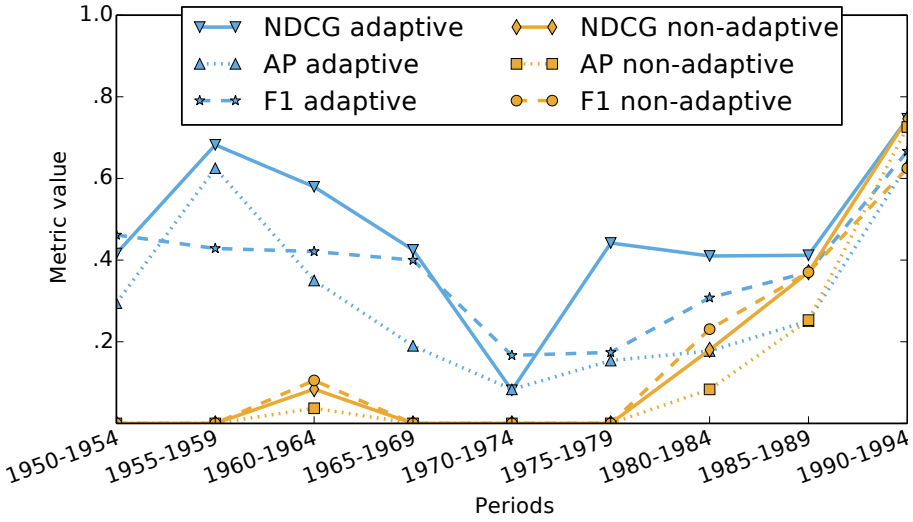


Figure 4.3: Comparison of results between the adaptive and non-adaptive run for the seed words "cd, compact disc." Direction is backwards in time.

Adaptive vs. non-adaptive

The non-adaptive method outperforms the baseline by itself. The adaptive method only does so in terms of F_1. As is clear from Table 4.2, though, the adaptive method can add valuable information to the non-adaptive method. In this section we present a number of examples to illustrate the difference between the two. To highlight the difference, we only show examples of the non-hybrid runs in this section. These runs contributed to the results in the rows labeled 'adaptive' and 'non-adaptive' in Table 4.2.

In Figure 4.3 the results are displayed for the non-adaptive run and the adaptive run for the seed words "cd, compactdisc." The direction for this example is backward, i.e., we start with the seed words in the 1990–1994 period and go backward in time.

[4]Note that due to macro-averaging, the macro-F_1 scores can and do end up lower than the individual macro-precision and macro-recall scores.

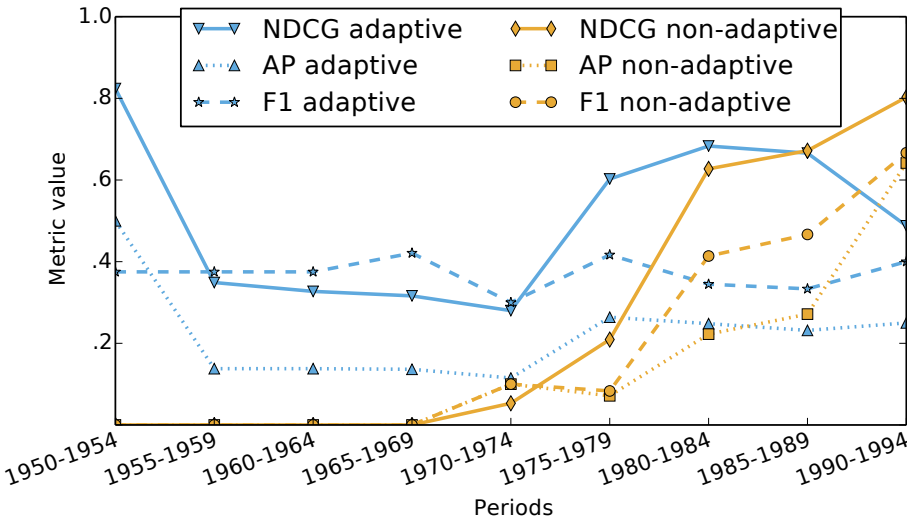


Figure 4.4: Comparison of results between the adaptive and non-adaptive run for the seed word "holocaust." Direction is backward in time.

As we can clearly see from the figure, the performance of the non-adaptive run quickly degrades over time (recall that we are going backward in time). Interestingly, the adaptive run, after a glitch in the 1970–1974 period, manages to pick up performance again in the time periods in the 1950s and 1960s. This indicates that the network approach, in which a network of related terms is promoted, can be beneficial.

We see a similar pattern in the results for the seed word "holocaust" in Figure 4.4. Again, we are going backward in time for this example. The performance of the non-adaptive run steadily degrades as we go back in time. This can be explained by the fact that the word "holocaust" barely occurs in the corpus prior to 1978.[5] The term was introduced in Dutch discourse by an American television series by that name. Initially, the term was used primarily to refer to the series, but gradually it became a more general term that now means the same as it does in English.

In Figure 4.5 the results are displayed for the seed word "multinational." The word "multinational" rarely occurs in the 1950s and 1960s in the Dutch digitized newspapers.[6] This is clearly reflected in Figure 4.5 and both the adaptive and the non-adaptive method suffer from this. Close inspection of the documents in which the word does occur in this period reveal that it is used in a political context (where it means international) rather than in a business context as later on. Importantly, the adaptive run is able to recover from its drop in performance, while the non-adaptive run is unable to do so, and keeps getting zero performance.

The examples in this section clearly show the limitations of non-adaptive approach that only follows a static set of words and the words related to them over time. If the words in the seed set do not exist in the period of interest (as in the "cd" example), change in meaning (the "multinational" example), or are not used throughout the entire period of interest (the "holocaust" example), a static approach will always fail.

[5] See: `http://kbkranten.politicalmashup.nl/#q/holocaust`
[6] See: `http://kbkranten.politicalmashup.nl/#q/multinational`

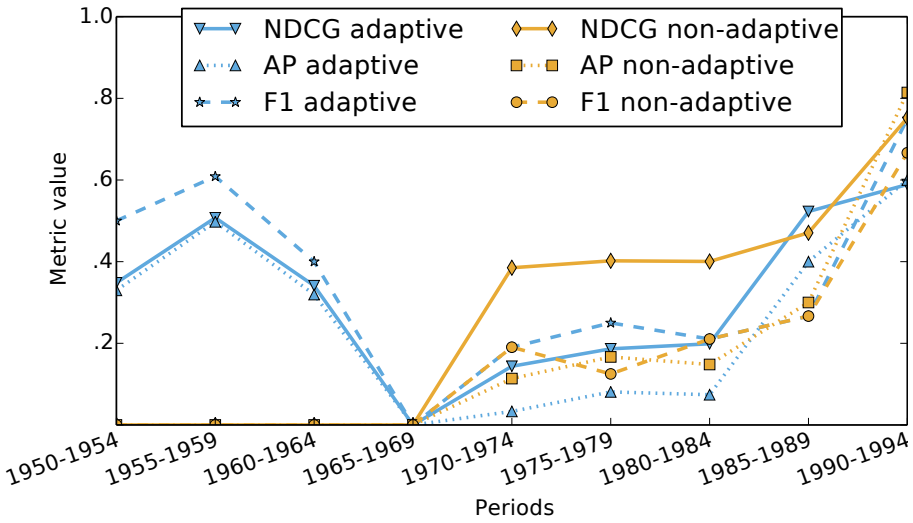


Figure 4.5: Comparison of results between the adaptive and non-adaptive run for the seed word "multinational." Direction is backward in time.

Table 4.3: Results for adaptive and non-adaptive method on a-historical seed sets only

Method	F_1	NDCG	MAP
adaptive	0.395	0.849	0.254
non-adaptive	0.387	0.872	0.254

Overgeneration

As discussed in §4.3.1 the evaluation set contains 5 a-historical seed term sets to check for overgenerating. In Table 4.3 we display the results on the a-historical subset of the ground truth seed sets, based on the same parameter settings used for Table 4.2.

As we can observe from Table 4.3 the results between the adaptive and non-adaptive runs are comparable. None of the differences is statistically significant for $\alpha = 0.05$ for a two-tailed paired t-test. We conclude from these results that our adaptive method for generating shifting vocabularies over time does not overgenerate. That is, if no changes occur in a vocabulary concerning a particular topic, none are in fact picked up by the adaptive method.

4.4.2 Parameter analysis

To answer research question RQ1.2 we analyze the effect of the parameters of the generation algorithm and the aggregation algorithm. For the generation algorithm the parameters are the length of the sliding time window, minimal semantic distance ς and the method of computing degree centrality. For the aggregation algorithm we have one parameter, the vocabulary weighting function.

Length of sliding time windows The length of the sliding time windows affects both the adaptive method and the non-adaptive method.

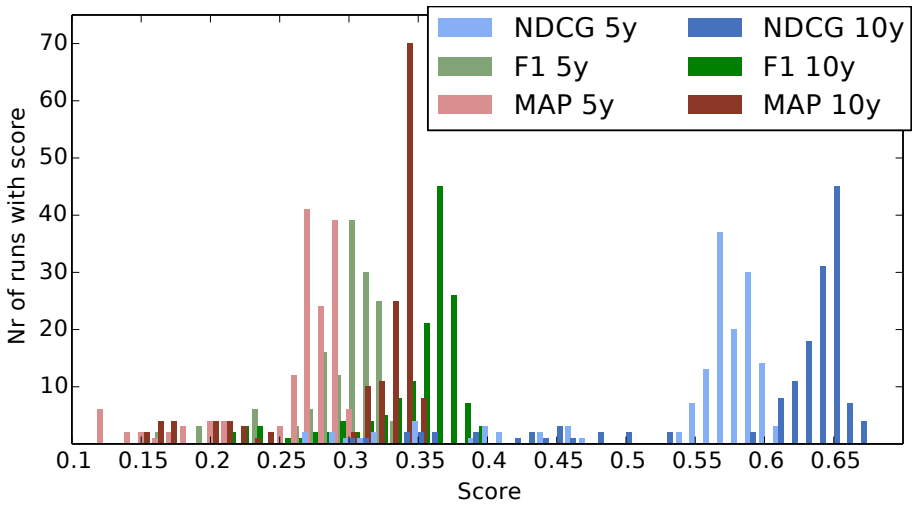


Figure 4.6: Comparison of results per metric, grouped by time window length.

Table 4.4: Results of a two-tailed paired t-test between the performance of all results per window length, paired by parameter setting.

Metric	t-statistic	p-value
NDCG	-20.8	5.34×10^{-19}
MAP	-24.0	1.08×10^{-20}
F_1	-19.6	2.73×10^{-18}

In Figure 4.6 performance of all runs — adaptive, non-adaptive and hybrid, all parameter settings — is plotted, grouped by window length. As is clear from the figure, using 10 year sliding windows yields better results in a vast majority of cases, for all metrics. In Table 4.4 the t-statistics and p-values are listed per metric for a paired t-test between the results per window length (the results are paired per parameter setting).

From these findings we conclude that using a longer time window to train a semantic model yields better performance for our current task, which supports the claim made in [118] that more training data yields better semantic models. Do note, though, that, due to the adaptive nature of our task, we can not use arbitrarily long time windows, as the changes in meaning and vocabulary we are interested in might go unnoticed that way.

Minimum distance As discussed in §4.2.2 the ς parameter controls which related words are taken into account for constructing semantic networks. In Table 4.5 the results across different levels of ς are displayed for all methods of generating shifting vocabularies over time, that use the generation algorithm (the non-adaptive method only uses the aggregation algorithm). The results are consistently lower than the results in Table 4.2, regardless of the method. This clearly indicates that a value of $\varsigma = 0.65$ is to be preferred for all methods, adaptive, non-adaptive or hybrid.

Degree centrality Regarding the different ways of calculating degree centrality we observe a very consistent pattern: choosing in-degree always yields better results than

Table 4.5: Top results for different settings of minimum similarity ς, all other settings as in Table 4.2.

Method	ς	F_1	p	r	NDCG	MAP
hybrid (i = 1)	0.7	0.367	0.521	0.389	0.630	0.332
	0.6	0.376	0.530	0.398	0.637	0.338
hybrid (i = 2)	0.7	0.368	0.523	0.385	0.630	0.332
	0.6	0.378	0.534	0.395	0.636	0.338
hybrid (i = 3)	0.7	0.372	0.530	0.388	0.634	0.335
	0.6	0.370	0.526	0.385	0.632	0.332
hybrid (i = 4)	0.7	0.370	0.529	0.381	0.632	0.333
	0.6	0.365	0.521	0.376	0.627	0.329
hybrid (i = 5)	0.7	0.366	0.525	0.372	0.628	0.331
	0.6	0.358	0.515	0.365	0.622	0.323
adaptive	0.7	0.316	0.678	0.241	0.485	0.220
	0.6	0.292	0.442	0.273	0.442	0.206

choosing out-degree. The best performance with out-degree, in terms of F_1, other settings as in Table 4.2 is yielded by the hybrid method, with i = 1. It yields an F_1 of 0.370, NDCG of 0.632 and MAP of 0.333, all of which is lower than the scores of the best performing hybrid runs.

Putting weights on the edges consistently leads to performance superior to unweighted edges. The best performance, in terms of F_1, without weighted edges, and other settings as in Table 4.2 is yielded by the hybrid method with i = 1, which yields an F_1 of 0.369, NDCG of 0.632 and MAP of 0.333.

Vocabulary weighting function In case of the non/adaptive method, not weighting the vocabularies leads to a small increase in performance: F_1 0.368, NDCG 0.632 and MAP 0.333, regardless of the value for minimum similarity ς. These differences, however, are not statistically significant for α = 0.5 for a two-tailed paired t-test. Furthermore, for the hybrid method, applying weighting for generating vocabularies over time nearly always yields better results when $i > 1$. These findings suggest that weighting of vocabularies is beneficial for generating shifting vocabularies over time.

4.4.3 Error analysis

In 9 cases of the 21, merging adaptive and non-adaptive runs for the hybrid runs led to performance that was less than the best performing of the two. In this section we will discuss three such examples. Typically, the decrease in performance was small ($\sim 1\%$).

Table 4.6 shows the vocabulary output for the hybrid run (i = 3) with seed set "marxism" for the 1990–1994 period. The direction is forward in time. This means that we start with the concept of marxism in 1950 and follow it as time progresses. As we can see from the results, the adaptive run has picked up on related terms and has become too general (the concepts, though they are related, are mainstream socio-economical movements, ideologies and isms). Much more on-topic words, like, e.g., "leninism" and

Table 4.6: Results of the hybrid run (i = 3) for seed set "marxism," for the last time period (the direction in time is forward). Words occurring in the ground truth set are marked with a *.

Period	Vocabulary[7]
1990–1994	communism*, marxism*, capitalism, humanism, christianity, socialism*, imperialism, atheism, militarism *(in two different spelling variants)*

Table 4.7: Results of the hybrid run (i = 3) for seed set "hydrogen bomb" for the last time period (forward direction in time). Words occurring in the ground truth set are marked with a *.

Period	Vocabulary
1990–1994	launching facilities, rockets, ballistic, launching pads, nuclear warheads*, nuclear submarines, atomic warheads, nuclear payload*, multi-headed, bomber

"stalinism," which were used in the late 1990s in the newspaper corpus are picked up by the non-adaptive run.

We see a different pattern for the run with the seed set "hydrogen bomb" in Table 4.7. Here, the adaptive run nearly loses track of the nuclear weapons completely, and rather focusses on missiles.[8]

The examples in this section show that the adaptive method for monitoring shifting vocabularies over time can be susceptible to topic drift. It can loose specificity (the "marxism" example) or it can drift in the wrong direction (the "hydrogen bomb" example). In cases like this particularly, a combination with a more conservative, non-adaptive approach is beneficial.

4.5 Conclusions

As discussed in Chapter 1, understanding texts at a word level means understanding how words relate to each other semantically. Motivated by research in digital humanities, where evolving viewpoints in society are studied by examining public discourse, our goal in this chapter was to automatically detect which words people used in different periods in time to refer to a particular concept. To study this we introduced the task of ad hoc monitoring of vocabulary shifts over time. In particular, we aimed to answer:

RQ1 Given a corpus of time-stamped documents, a point in time and a small set of seed terms used to denote a concept in that corpus at the time specified, can we

[7] The original words are in Dutch, translations by the authors

[8] The term "atomic warheads" was not annotated as correct, even though it means the same as "nuclear warhead," because it was hardly ever used, while "nuclear warhead" was used abundantly.

infer from the corpus which terms are used in adjacent periods in time to denote the same underlying concept?

We presented several algorithms for monitoring vocabularies over time and performed systematic, intrinsic evaluation of their results. Our results show that our approach of combining an exploratory method of generating shifting vocabularies over time with a conservative approach consistently and significantly beats a baseline inspired by related research, and that it consistently performs better than the two approaches it combines.

Intrinsic evaluation of semantic methods is difficult. Constructing a manually labelled dataset as we did is costly and labour-intensive. We hope that disclosing the full evaluation set is beneficial to research in this area.

The output of our method of monitoring shifting vocabularies over time are word lists, specifically tailored towards a particular period in time. The lists could be used for time-aware query expansion, where the query expansion depends on the timestamps of documents in a corpus. While in our experiments in §4.3 we aimed to evaluate our method intrinsically, it would be interesting how our proposed method would perform when extrinsically evaluation based on time-aware dynamic query expansion.

The performance of an adaptive method for monitoring shifting vocabularies may degrade or improve over time. However, traditional evaluation metrics like NDCG or MAP are time-agnostic. Additional insights could be obtained when a time-aware evaluation metric, such as, e.g., proposed in [82] in the context of document filtering systems, would be applied to the present setting.

While we used in-degree and out-degree to measure saliency of terms in semantic graphs, additional graph-based measures, like PageRank [26], could be taken into account.

Having presented our research on text understanding at the level of words in this chapter, we now turn to the next section of this thesis: natural language understanding at short-text level.

Part II

Short texts

5 Short Text Similarity with Word Embeddings

5.1 Introduction

Determining semantic similarity between two texts is to find out whether two pieces of text have a similar meaning. Being able to do so successfully is beneficial in many settings in information retrieval like search [105], query suggestion [116], automatic summarization [6] and image finding [34].

Many approaches have been proposed for semantic matching that use lexical matching and linguistic analysis, next to semantic features. Methods for lexical matching aim to determine whether the words in two short texts look alike, e.g., in terms of edit distance [117], lexical overlap [75] or largest common substring [72]. While this might work for trivial cases, it is arguably not robust as it allows for simple mistakes. For example, the *US* would be closer to the *UK* this way, than it would be to the *States*. Features based on linguistic analysis, like dependency parses or syntactic trees, are often used for short text similarity [58, 139]. Linguistic tools such as parsers are commonly available these days for many languages, though the quality might vary between languages. However, not all texts are necessarily parseable (e.g., tweets) and high-quality parses might be expensive to compute at run time. More importantly still, relying on parse trees limits an approach to single sentences, while the work presented here, even though it is evaluated on sentences, incorporates no theoretical constraint restricting it to (syntactically well-formed) sentences. For semantic features, many approaches use external sources of structured semantic knowledge such as Wikipedia [13] or WordNet [13, 44, 45, 117, 134]. Wikipedia is structured around entities and as such is primarily of avail in settings where a focus on rather well-known persons and organizations can be assumed, such as, e.g., news articles. Such an assumption cannot always be made however. A drawback of using dictionaries or WordNet is that high-quality resources like these are not available for all languages, and proper names, domain-specific technical terms and slang tend to be underrepresented [3].

In this chapter we aim to make as few assumptions as possible. We aim for a generic model, that requires no prior knowledge of natural language (such as parse trees) and no external resources of structured semantic information. Specifically, we aim to answer the following research question:

RQ2 How can pre-trained word embeddings be used to calculate similarity between short texts, without relying on linguistic structure?

Recent developments in distributional semantics, in particular neural network-based approaches like [118, 124] only require a large amount of unlabelled text data. This data is used to create a, so-called, semantic space, in which terms are represented as vectors that are called *word embeddings* (cf. §2.1). The geometric properties of this space prove to be semantically and syntactically meaningful [35, 118, 119, 124]. That is, words that are semantically or syntactically similar tend to be close in the semantic space.

A challenge for applying word embeddings to the task of determining semantic similarity of short texts is going from word-level semantics to short-text-level semantics. This problem has been studied extensively over the past few years [8, 103, 139].

In this chapter we propose to go from word-level to short-text-level semantics by combining insights from methods based on external sources of semantic knowledge with word embeddings. In particular, we perform semantic matching between words in two short texts and use the matched terms to create a saliency-weighted semantic network. A novel feature of our approach is that an arbitrary number of word embedding sets can be incorporated, regardless of the corpus used for training, the underlying algorithm, its parameter settings or the dimensionality of the word vectors. We derive multiple types of meta-features from the comparison of the word vectors for short text pairs and from the vector means of their respective word embeddings, that have not been used for the task of short text similarity matching before.

We show on a publicly available test collection that our generic method, that does not rely on external sources of structural semantic knowledge, outperforms baseline methods that work under the same conditions and, moreover, outperforms all methods, to our knowledge, that do use external knowledge bases and that have been evaluated on this dataset.

We present our method for short text similarity in §5.2. The experiments and results are detailed in §5.3 and §5.4.

5.2 Short text similarity with semantics only

To calculate semantic similarity between two short texts we use a supervised machine learning approach. Algorithm 3 shows the pseudocode of the training phase. The training data for the supervised step consists of sentence pairs and associated labels that represent the semantic similarity between the two sentences. Multiple sets of word embeddings can be leveraged, possibly derived from different corpora, with different (hyper)parameter settings or with different algorithms. Every sentence pair in the training data is represented by a set of features. A list of functions that generate features from a set of word embeddings and a sentence pair is required. We detail the different kinds of features below in §5.2.1 and §5.2.2.

At training time, we range over all sentences (Algorithm 3, line 2), all sets of word embeddings (line 4) and feature extraction functions (line 5) to compile a feature vector per sentence pair (line 6). The feature vectors are stored in a matrix (line 9). We train a supervised learning method from the features and the labels of the training examples

Input : List of sentence pairs $((s_{1,1}, s_{1,2}), (s_{2,1}, s_{2,2}), \ldots, (s_{n,1}, s_{n,2}))$
Input : List of associated labels $L = [l_1, l_2, l_3, \ldots, l_n]$
Required : Sets of word embeddings $[WE_1, WE_2, ..., WE_m]$
Required : Multiple feature extractors $[fe_1, fe_2, \ldots, fe_l]$
Output : A trained prediction model M

```
1  F = empty feature matrix;
2  for i ← 1 to n do
3  |   f⃗ = <>;
4  |   for j ← 1 to m do
5  |   |   for k ← 1 to l do
6  |   |   |   f⃗ = concat(f⃗, fe_k((s_{i,1}, s_{i,2}), WE_j));
7  |   |   end
8  |   end
9  |   F[i] ← f⃗;
10 end
11 M = trainModel(F, L);
```

Algorithm 3: Pseudocode of the feature generation step of our method for semantic similarity of short texts

(line 11). As the labels in the evaluation set that we use are binary, we build a classifier. At testing time, features are generated for the sentence pairs in the test set in a similar fashion as in the training phase, and a final prediction is made with the classifier trained in the training step.

A convenient property of our method of computing semantic textual similarity for short texts, and one which we leverage in our experiments as detailed in §5.3, is that different sets of word embeddings can be combined, regardless of the dimensionality of the word vectors, the parameters that were used at construction time, or the algorithms that were used to generate them.

As we are interested in the performance of different feature types, we carry out experiments per feature type and with various combinations. See §5.3 for further details on the experimental setup.

In the next section we describe the various types of features we derive from the word embeddings. In §5.3.2 we detail how we obtain the word embeddings themselves.

5.2.1 From word-level semantics to short-text-level semantics

The meaning of longer pieces of text (containing multiple terms) can be captured by taking the mean of the individual term vectors.[1] This approach is taken, next to other approaches, in, e.g., [12, 70, 141]. It works surprisingly well, and we use several features based on vector means, described below. Means, or sums, however, are rather poor ways of describing the distribution of word embeddings across a semantic space. It would be desirable to capture more properties of the two texts, especially with respect to the terms that do or do not match. We will first turn to our algorithm for constructing

[1] This is sometimes referred to as vector BOW approach

saliency-weighted semantic networks, which aims to capture this intuition. After that, in §5.2.2, we discuss features based on the mean vectors.

Saliency-weighted semantic network

In Figure 5.1 the word embeddings of two short texts are represented as dots in a two-dimensional space. As can be observed from the picture, the two texts have terms that are close to each other (at the top and bottom in the figure), while the ones at the far left and right have no counterpart in their immediate vicinity. Regardless of this discrepancy, the means of the two are close to one another. The fact that both texts have a term unlike any term in the other text is not well represented by the means. A classifier, however, can benefit from more elaborate information about the distribution of word embeddings across the semantic space.

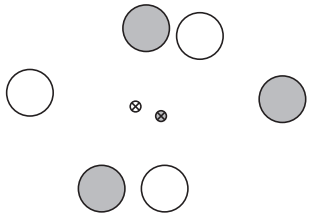


Figure 5.1: Hypothetical example — two-dimensional representation of the word embeddings for two short texts (each consisting of three terms), represented as transparent and opaque dots respectively. The corresponding means of the two sets of embeddings are depicted as $\otimes$.

We want a way of taking into account the distribution of terms in one short text in the semantic space compared to distribution of terms in another text. Of course, not all terms are equally important. Common terms (like determiners) do not contribute as much to the meaning of a text as less frequent words do. Inverted document frequency (idf) is often used to implement this notion. Idf is usually combined with term frequency, e.g., in the BM25 algorithm. As BM25 [129] has proven to be a robust method of combining query and document texts in information retrieval (IR) research we derive our idf weighting scheme, used in comparing two short texts, from it. Our function for calculating semantic text similarity (sts) is:

$$f_{sts}(s_l, s_s) = \sum_{w \in s_l} \text{IDF}(w) \cdot \frac{sem(w, s_s) \cdot (k_1 + 1)}{sem(w, s_s) + k_1 \cdot (1 - b + b \cdot \frac{|s_s|}{\text{avgsl}})} \tag{5.1}$$

Here, s_l is the longest text of the two, s_s is the shortest and avgsl is the average sentence length in the training corpus.

The semantic similarity of term w with respect to short text s is represented by $sem(w, s)$:

$$sem(w, s) = \max_{w' \in s} f_{sem}(w, w'). \tag{5.2}$$

The function f_{sem} returns the semantic similarity between two terms. As terms are represented as vectors in our case, a natural choice for f_{sem}, which we use in our experiments in §5.3, is the cosine similarity between the two vectors.

As is apparent from Equation 5.1, we always take the longest short text of the two as a reference when calculating f_{sts}. We do so for two reasons. Firstly, we want f_{sts} to be symmetrical. Calculating the semantic similarity between two short texts should yield the same score regardless of their order. Secondly, the reason why the longest of the two short texts is summed over is that we do not want terms to be overlooked. Suppose we have two texts, where one consists of a subset of terms contained in the other. If the shortest text would be taken as a reference this would lead to a perfect score. However, if we take the longest text as a reference, the incongruity between the texts does have its bearing on the score, as desired.

We should note that, although Equation 5.1 bears a superficial resemblance to the BM25 formula, it in fact models something completely different. We borrow the b and k_1 parameters that have a smoothing effect, together with the length normalization: the average sentence length, $avgsl$, in our case. The key difference, however, lies in the introduction of semantic similarity term in the formula. Where a tf*idf weighting scheme relies on literal matches between the query and documents it matches, we are, in the present setting, interested in particular in semantic matches. By using Equation 5.2 for calculating semantic similarity, the maximum similarity of terms in s_s is taken into account for every term in s_l.

One interpretation of f_{sts} is that it allows for non-literal, semantic matching. As noted above, in a tf*idf weighting scheme, terms only contribute to the score if they match perfectly. In f_{sts} all terms contribute, with the semantically most related ones contributing most.

An alternative interpretation of f_{sts} is as a word alignment method. As a $\max$ is being computed in Equation 5.2 over all words in a sentence, semantically close words are aligned to one another. In this way f_{sts} bears similarity to other alignment approaches such as [12, 58, 113].

Yet another alternative view is that f_{sts} applies saliency weighting to a semantic network. If we interpret the dots in Figure 5.1 as vertices of a graph, the $\max$ in Equation 5.2 draws edges between the vertices and weights them according to Equation 5.1. As a result, a mismatch between two terms such as the far left and right ones in Figure 5.1 is of little consequence if both have a low IDF score (e.g., they are function words). If they are salient however, the mismatch has a larger impact.

As can be seen from Equation 5.1 the f_{sts} score is a sum over $|s_l|$ terms. However, rather than giving the overall score to the final learning algorithm, we want to capture more information about the way the score is composed. Therefore, we make bins of its summands and normalize by the number of summands, so the value for every bin represents the percentage of summands in Equation 5.1 between the minimum and maximum values for that bin.

Unweighted semantic network

To convey as much information as possible to the final classifier we also construct an unweighted semantic network. For a short text pair (s_1, s_2), we compute the cosine similarities in the semantic space between all terms in short text s_1 and all terms in s_2. This gives us a matrix of similarities between the terms in s_1 and s_2. From this matrix we compute two sets of features.

Firstly, we take all similarities and bin them. If we think of the word embeddings as nodes in a graph this would correspond to an fully connected, unweighted, bipartite graph. In Figure 5.1 this would be represented by connecting every opaque dot to every transparent dot.

Secondly, the maximum similarity for every word is computed, and bins are made of these maximum values. In this way, small distances between words (such as the top and bottom ones in Figure 5.1) end up in the same bin, while outliers (the ones at the far left and right) end up in a separate bin.

5.2.2 Text level features

Distance between vectors means

As noted above in §5.2.1, a standard way of combining word embeddings to capture the meaning of longer pieces of text is to take the mean of the individual term vectors. This aggregation over terms gives us one vector per sentence. We calculate both the cosine similarity and the Euclidean distance between the vectors for every sentence pair in the test set.

Bins of dimensions

The cosine similarity between two vectors can be interpreted as an aggregation over the differences per dimension. As such, it does not capture all information about the similarities or differences between the two vectors. For example, taking the cosine similarity between two vectors that are highly similar in many dimensions and quite different in few, could lead to the same result as taking the cosine similarity between two vectors that differ slightly in all dimensions. Intuitively though, these are two different situations. In order to capture this intuition, we make bins of the number of dimensions in the mean vector of s_1 and the mean vector of s_2 that match within certain limits. See §5.3.3 for the exact values.

5.3 Experimental setup

To answer RQ2, the focus of our experiments is to determine how our method, which relies solely on semantic features, compares to other methods that work under the same conditions, and to methods that do rely on external sources of structured semantic knowledge, linguistic tools and handcrafted rules. To do so, we perform experiments on the MSR Paraphrase Corpus [38, 126], the evaluation set most commonly used for this purpose.

As described in the previous section, we compute features from word embeddings, which are obtained from large amounts of unlabelled data. A practical feature of word embeddings is that vectors computed on a large corpus can be made available, without the necessity of disclosing the entire training corpus as well (which can be problematic due to copyright issues). In our experiments we compute features from four publicly available sets of word embeddings (see §5.3.2 and §5.3.2 for more details). As the word vectors are not trained by ourselves, we refer to them as Out-of-the-Box (OoB).

We distinguish between two feature sets. The *saliency-weighted semantic network* features capture information about the similarity of the distribution of word vectors in the two sentences (§5.2.1). The *unweighted* features are calculated from the unweighted semantic network (§5.2.1) and the means of the word embeddings of both sentences (§5.2.2). Our hypothesis is that saliency-weighted semantic networks can add valuable features for a classifier that learns to predict semantic similarities between short texts. To verify this hypothesis we perform experiments without the features based on the saliency-weighted semantic networks, and with the saliency-weighted semantic network features added.

Additionally, as there are many parameters that have an impact on the word embeddings we use to construct features, we want to investigate which settings lead to word embeddings best suited for our approach. As it is not possible to do this with out-of-the-box vectors, we construct our own vectors from a publicly available text corpus (see §5.3.2 for details). As we use the features derived from these sets of word embeddings supplementary to the OoB features, we refer to them as *auxiliary*.

For the experiments with features derived from the OoB vector sets, the only hyper-parameters of our model are the regularization parameters of the learning algorithm. We choose their optimal setting by cross validating on the training data with folds of 10% of the examples. The experiments including the auxiliary vectors are aimed at demonstrating the potential of our method and the effect of the parameter settings. Therefore, we show the best results obtained across all settings and discuss the individual parameter settings in detail.

5.3.1 Learning algorithm

As discussed in §5.2 we use the features described above to represent sentence pairs in the training material and we train a supervised learning algorithm on these features. As the MSR Paraphrase Corpus is annotated with binary labels (see §5.3.6) we use a classifier for prediction. In particular, we use Support Vector Classifier (SVC) with a Radial Basis Function (rbf) kernel because the feature space is not necessarily linear.

5.3.2 Word embeddings

For ease of comparison with other approaches using word embeddings, we use four sets of vectors that are publicly available (two word2vec sets and two GloVe sets) which we refer to as Out-of-the-Box sets (OoB). Additionally we train our own word vectors, both with word2vec and GloVe, and perform runs with different settings for both algorithms (the auxiliary vectors).

Once word embeddings have been trained on a corpus, there is no way to fold terms that were not observed during training into the semantic space. One way of dealing with these out-of-vocabulary (OOV) words when calculating the features described below is to simply ignore them. However, it is possible that important semantic information is present specifically in these new words. For example, names of persons or organizations occurring in a test set, which are likely to be semantically relevant, might have been absent from the training data. Therefore, following, e.g., [88], we map OOV words to random vectors, while remembering which OOV word maps to which random vector.

The intuition behind this simple scheme is the following. If two texts are being compared in which two different OOV names appear, this incongruity would go unnoticed when the OOV terms would be ignored. Likewise, if the same OOV term is observed in two texts being compared, this should contribute to the similarity score between the two (instead of being silently ignored).

OoB: Word2vec

Mikolov et al. [119] experiment with several settings of the word2vec algorithm to produce the highest quality word embeddings. The resulting vectors have been made publicly available.[2] The vectors are 300-dimensional and were trained on a corpus of about 100 billion words.

Baroni et al. [14] compare word2vec word embeddings to traditional distributional semantics approaches. The best performing vectors were released by the authors.[3] The vectors are 400-dimensional, a 5-word context window was used, with 10 negative samples and subsampling.

OoB: GloVe

In [124] an algorithm is proposed for deriving word embeddings optimized especially for word analogy and similarity tasks. As the GloVe algorithm differs from the word2vec algorithm, it is interesting to see whether and how GloVe word embeddings behave differently from word2vec vectors when applied to the task of short text similarity. In our experiments we use two sets of publicly available GloVe vectors. Both sets are 300-dimensional. The word vectors of the first set were trained on a very large corpus of 840 billion tokens while the other set was trained on a 42 billion token corpus.[4]

Auxiliary word embeddings

As we are interested in the utility of the vectors and what settings work best for the task of predicting short text similarity, we calculate auxiliary word embeddings both with the word2vec algorithm and with GloVe. We train word embeddings on a publicly available data set released by INEX.[5] The corpus contains 1.2 billion tokens.

The word2vec algorithm has several parameters: the architecture (CBOW or Skip-gram), word sampling threshold, whether or not to apply hierarchical softmax and the number of negative examples. Preliminary experiments indicated that a sampling threshold of 10^{-5} is most robust across settings. We use the default window width of 5 and vector dimensionality of 300.

For the auxiliary GloVe vectors we use the same dimensionality of 300. We set the number of training iterations to 100 as is suggested in [124] for training vectors of dimension 300 and up. There are two parameters in particular to experiment with: (1) the exponent of the weighting function used in the cost function, which we set to

[2] See `https://code.google.com/p/word2vec/`

[3] See `http://clic.cimec.unitn.it/composes/semantic-vectors.html`

[4] Both sets of vectors can be downloaded from `http://nlp.stanford.edu/projects/glove/`

[5] The data consists of an English Wikipedia dump from November 2012. It was released as a test collection for the INEX 2013 tweet contextualization track [15]

any of [0.1, 0.5, 0.75, 0.9], and (2) the cut-off in the weighting function, which we set to any of [10, 50, 100, 500, 1000].

Preprocessing of the corpus consists of tokenization with the NLTK sentence splitter and token splitter [19] with additional removal of non-ascii quotes and non-word characters. No stemming or stopping is carried out. All text is lowercased.

5.3.3 Parameter settings

As discussed in §5.2.1 and §5.2.2 a binning approach is used for most features. We use three bins in most cases, where one bin is meant to capture highly similar values, one bin is for the medium values and the third bin is for very dissimilar values. The values were obtained by examining the raw features for the training material. For features calculated from the saliency-weighted semantic network, the values are 0–0.15, 0.15–0.4, 0.4–∞. For the unweighted semantic network features the values are -1–0.45, 0.45–0.8, 0.8–∞ (the same values are used for when all similarities are taken into account, as when only the maximum similarities are considered). For the bins of dimensions, preliminary experiments showed that a four-bin approach, with two bins for similar and highly similar values worked slightly better than a three-bin approach. We use values $-\infty$–0.001, 0.001–0.01, 0.01–0.02, 0.02–∞.

As an extensive tuning of the parameters k_1 and b is beyond the scope of our research at present we use the default settings of k_1 = 1.2 and b = 0.75 when computing f_{sts} in our experiments. The IDF values were calculated from the INEX data set described above.

5.3.4 Feature sets

All feature sets are calculated per set of word embeddings. Hence, for 3 saliency-weighted semantic network bins, 2 $\times$ 3 unweighted semantic network bins, 2 distances and 4 dimensional bins, we have 15 features per set of word embeddings, and 60 features in total per sentence pair, when, e.g., the 4 OoB sets are used.

5.3.5 Baselines

As discussed in §3.2.1 the systems for detecting short text similarity as described in [70, 72, 138] are natural baselines to our method as they work under the same conditions, i.e., no external sources of structured semantic knowledge are used and no prior knowledge of natural language (such as parse trees) is required.

5.3.6 Evaluation

We first describe the evaluation set used in our experiments. Then we discuss the associated evaluation metrics.

Evaluation sets

We use the Microsoft Research Paraphrase Corpus data set [38, 126] in our primary experiments in §5.4 as it is commonly used for evaluation in short text similarity

tasks [8, 44, 70, 72, 117, 138]. The set consists of sentence pairs judged for semantic similarity on a binary scale. The annotator guidelines allowed for an interpretation of semantic similarity that went beyond strict semantic identity, as using the latter notion would yield only trivial examples. The set consists of 5801 sentence pairs in total, divided in a training set of 4076 and a test set of 1725 examples.

Other evaluation sets Li et al. [106] present a data set comprising 65 pairs of dictionary glosses extracted from two sources. The set is used for evaluation in, e.g., [45, 72]. We omit this set in our experiment because of its limited size.

The task of semantic textual similarity was part of the SemEval 2012 and SemEval 2013 campaigns [2, 3]. Part of the MSR Paraphrase Corpus is incorporated in SemEval 2012 dataset.

The SemEval data is impractical for evaluation in a supervised learning setting with a substantial number of features, as the training data is limited (maximally 750 training examples per subset in the SemEval 2012 data) which leads to overfitting. Additionally, more recent work in semantic textual similarity is evaluated on the MSR Paraphrase corpus. For these two reasons, we evaluate our methods for calculating semantic text similarity on the latter in our main experiments in §5.4. We present results on the SemEval datasest in the additional experiments in §5.5.

Evaluation metrics

As the MSR Paraphrase Corpus has binary annotations, accuracy is the metric most often applied, together usually, with precision, recall and F_1 [8, 44, 70, 117].

5.4 Results and analysis

In this section we present the results of our main experiments. To answer RQ2, we are interested in answering two questions. Firstly, we want to see whether our method performs better than the baseline methods that work under the same conditions. Secondly, we want to know whether a semantics-only approach, without access to external sources of knowledge, can yield results comparable to the state-of-the-art methods that do use external semantic knowledge bases and/or features based on computationally more involved processes as syntactic parsing.

In Table 5.1 results of our experiments are listed. For convenience, the results from the baseline methods, as reported in the literature, are displayed in the top three rows.[6] The rows marked ‘unweighted use all features described above, but for the features based on the saliency-weighted semantic network. The rows marked ‘unweighted + swsn’ use both the unweighted features and the saliency-weighted semantic network features.

[6]No precision and recall numbers were reported in [70].

Table 5.1: Results on the MSR Paraphrase Corpus set. The rows marked 'unweighted display results for runs based on all features, but for the saliency-weighted semantic network features. The rows marked 'unweighted + swsn' display results for runs that had features based on saliency-weighted semantic network added as well. Results marked† are significantly different from the best performing OoB run (two-tailed paired t-test, p-value < 0.007).

Baseline methods		Accuracy	Precision	Recall	F_1
Convolutional NNs [70]		0.699	–	–	0.809
VSM [138]		0.710	0.710	0.954	0.814
Corpus-based PMI [72]		0.726	0.747	0.891	0.813
Our method	**Features**	**Accuracy**	**Precision**	**Recall**	F_1
OoB	unweighted	0.746	0.768	0.882	0.822
OoB	unweighted + swsn	0.751	0.768	0.896	0.827
OoB + aux w2v	unweighted	0.754	0.770	0.897	0.829
OoB + aux w2v	unweighted + swsn	0.757	0.775	0.894	0.830
OoB + aux Glv	unweighted	0.756	0.774	0.894	0.830
OoB + aux Glv	unweighted + swsn	0.758	0.771	0.907	0.833
OoB + both aux	unweighted	0.762†	0.780†	0.893†	0.833†
OoB + both aux	unweighted + swsn	0.766†	0.781†	0.906†	0.839†

5.4.1 Using out-of-the-box vectors

For the rows marked 'OoB' only out-of-the-box word embeddings were used of the four sets described in §5.3.2 and §5.3.2. The settings for the regularization parameters of the classifier are determined by cross validating on the training set. For the experiment with only unweighted features, the hyper-parameter settings are C = 10^8 and gamma = 10^{-5}. For the experiment including the saliency-weighted semantic network features we have C = 10^6 and gamma = 10^{-4}.

Table 5.1 shows that the result when using only publicly available, out-of-the-box word vectors (the rows marked OoB), surpass all the baselines.[7]

An important observation is that the best scoring approach on this data set using WordNet-based features, to our knowledge, reports an accuracy of 0.741 and an F_1 score of 0.824 [44]. As we can see, our generic approach with only publicly available vectors, without any tuning or optimization, outperforms this method. This is an important finding, as it shows that for computing semantic similarity, the labour-intensive construction of a rich semantic knowledge source such as WordNet is not a necessity. As an aside, our finding supports a claim made in [12] in a different context of matching texts of different lengths, "that traditional knowledge-based features are cornered by novel corpus-based word meaning representations."

[7] We cannot calculate statistical significance between our results and the baseline results as for the appropriate test — a matched-pairs t-test — we need to compare our output to the outputs of the baseline systems, which are not publicly available (only the aggregate results are).

5.4.2 Using auxiliary vectors

To show the potential of our method we report the results when, next to the OoB vectors, the auxiliary vectors — generated from the embeddings trained on the INEX data as described in §5.3.2 — are used. The bottom rows in Table 5.1 show the best results obtained with these vectors.

The results of the experiments with the auxiliary vectors are consistently better than the baselines and the results with only OoB vectors, both in terms of accuracy and F_1.

If we compare the results of our method that only uses semantic features, to the current state-of-the-art methods, which rely on linguistic analysis and handcrafted features, we observe that when optimal settings are used, our method can outperform the tree kernel approach described in [8]. This method uses features derived from dependency parses, and yields an accuracy of 0.753 on this data set (no precision, recall and F_1 score are reported). Furthermore, our top-performing run, the bottom row in Table 5.1, shows results comparable results presented in [139], based on dynamic pooling and unfolding recursive auto-encoders trained on parse trees — that has an accuracy of 0.768 and F_1 of 0.836. Our top-performing run does slightly better in terms of F_1 and slightly worse with respect to the accuracy. It is important to note, however, that the results in [139] are only achieved when, next to the general neural network-based method, several handcrafted features are added, which are designed especially for the evaluation set at hand (the features are primarily dealing with representation of numbers). Interestingly, our method performs better than the neural network-based approach in [139] when the latter is run without the test-set-specific handcrafted features, in which case it yields an accuracy of 0.726.

The best performance on the MSR Paraphrase Corpus, to our knowledge, is presented in [73]. Matrix decomposition is performed on a co-occurrence matrix, and saliency weighting is applied, where the saliency weight per word or n-gram is optimized on the training data. A dependency parser is used to generate the ngrams. The best performance is obtained by performing matrix decomposition on the training and test data combined. But even when only the training set is used, an accuracy of 0.786 and 0.846 F_1 is attained, when no additional hand-crafted features are used. This result is better than ours in terms of accuracy, while in terms of F_1 the scores are comparable.

The auxiliary word2vec vectors and GloVe vectors, when added separately, yield better performance. This is particularly noteworthy as it shows that high-quality word embeddings can be produced for the present setting by both algorithms, even when the corpus used for training was substantially smaller than what is commonly used (namely $\sim$1B tokens, against 3B and 100B tokens for the OoB word2vec sets and 42B and 840B for the OoB GloVe vectors).

When both the auxiliary word2vec and GloVe vectors are added to the OoB vectors we see another increase in performance specifically in terms of precision — the rows marked ‘OoB + both aux’. These results are significantly different from the OoB run with both feature sets. The results in the ‘OoB + both aux’ rows also surpass the results in the ‘OoB + aux w2v’ and ‘OoB + aux Glv’ separately. This is particularly interesting, as it indicates that, while the performance of both vectors sets on their own is comparable, the two models capture different semantic information.

Finally, an overall observation from Table 5.1 is that adding the saliency-weighted

semantic network features consistently yields better performance. We note that this goes against an observation in [44]: *"Since experiments with document specificity weightings (such as tf-idf) had shown that using these factors actually reduced performance no such weighting factor was used here."*

Hyperparameter analysis

An important observation we make from studying the performance of our method for combining word embeddings for a short text similarity task, is that the results do not vary greatly between different settings of hyperparameters, except for the regularization parameters of the classifier. To illustrate, for the 'OoB + both aux' setting, the worst performance in terms of accuracy across hyperparameters, with optimal regularization parameters was 0.730, which is worse than the performance with only OoB vectors, but above baseline performance. For the auxiliary word2vec vectors, negative sampling seems to be beneficial in general, with a value of 10 being a robust choice. Both the choice of architecture (CBOW or Skip-gram) and applying hierarchical softmax or no seems to be of little consequence.

For the GloVe vectors different values for the exponent of the weighting function and the cut-off in the weighting function were used. The moderate values (0.5, 0.75 for the exponent, and 50 or 100 for the maximum cut-off) yielded the best results.

Lastly, the regularization parameters (C and gamma) of the classifier are hyperparameters of our model. We use large values for C (in the range of 10^6–10^9 depending on the number of features) and small values for gamma (10^{-4}, 10^{-5}). Not surprisingly, the setting of these parameters in particular has substantial repercussions on the final performance. To illustrate again, the worst performance with optimal features but across regularization parameters was 0.690, which is lower than the lowest baseline. The worst overall performance (worst features and worst regularization) is roughly equal, at 0.685, which suggests that the harm is primarily caused by suboptimal regularization.

The findings reported here indicates that both algorithms for generating word embeddings, word2vec and GloVe, are robust across reasonable parameter settings. While it is important to find the optimal combination of all parameters, the settings for the word embeddings matter less than regularizing the learning algorithm.

Feature importance

In addition to studying the effect of the different sets of word embeddings as discussed above, it is interesting to see how different sets of features affect the performance. To analyze the effect per feature set we perform an ablation study, where we leave out a set of features for all word embedding sets we calculate features from.[8]

The results for leaving out different feature sets, sorted by accuracy, are shown in Table 5.2. All other settings (the word2vec and GloVe parameters for the auxiliary vectors and the regularization parameters for the classifier) are identical to the ones used for the top performing run in Table 5.1 (bottom row).

[8]Note that it is not possible to use the feature weights as a proxy for feature importance, as this only works for linear kernel functions, and we use a classifier with an RBF kernel.

Table 5.2: Ablation study results.

Omitted feature set	Accuracy	Precision	Recall	F_1
max unweighted sn bins	0.739	0.766	0.874	0.817
swsn bins	0.741	0.768	0.874	0.818
dimension bins	0.746	0.767	0.886	0.822
all unweighted sn bins	0.747	0.763	0.898	0.825
distances	0.759	0.778	0.892	0.831

As can be seen from Table 5.2 leaving out the word alignment methods, weighted by saliency or not, has the most dramatic effect on performance. This indicates that aligning words is a successful strategy for determining semantic similarity between short texts.

An interesting observation is that leaving out the distance features has the least effect on performance. As noted above, these features measure the distance between the means of the word vectors in both sentences, and are a default method for going from word-level to sentence-level, applied, next to other methods, in e.g., [12, 70, 141]. Table 5.2 does suggest that binning the differences between dimensions of the mean vectors, as proposed in this paper, increases the gain obtained from them.

5.4.3 Error analysis

To see whether our method of computing semantic textual similarity for short texts is biased we perform an error analysis concerning two important attributes of the test data: sentence length and lexical overlap.

Performance across sentence length

Figure 5.2 shows an overview of the results of our experiments divided by sentence length. As expected, sentences that are alike in in terms of length are easier to perform well on, as reflected in the figure by the large bulge at the left side of the scale for the true positives. The hump for true negatives is less pronounced, which is easily explained by a lower frequency of negative examples in the test set.

One observation from Figure 5.2 is that the number of false negatives is rather constant in the left half of the figure, which means that it increases relatively with the difference in sentence length, while this is not the case for false positives. This means that the classifier has a tendency to predict semantic dissimilarity when the two input texts differ in length substantially.

Most importantly though, Figure 5.2 shows that the classifier always predicts the correct label in the majority of cases, regardless of the difference in sentence length.

Performance across levels of lexical overlap

As our saliency-weighted semantic network features perform semantic, rather than lexical, word pairings, it is interesting to see how our method performs across different levels of lexical (i.e., *literal*) overlap between the sentence pairs in our test collection.

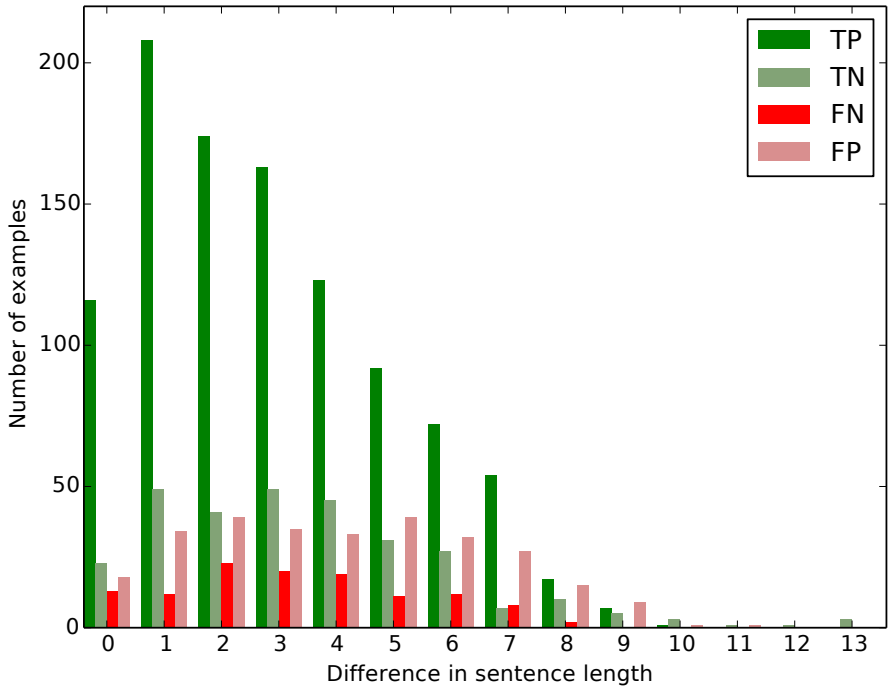


Figure 5.2: Results 'OoB + both aux – unweighted + swsn' run divided by difference in sentence length (measured in words). TP: true positives, TN: True negatives, FP: false positives, FN: false negatives.

In Figure 5.3 we show an overview of the results across different levels of lexical overlap of our best performing run on the MSR Paraphrase Corpus (OoB + both aux – unweighted + swsn; see bottom row in Table 5.1). We can clearly see four different distributions of results, where, e.g., the TP results peak at 70–80% and the TN results peak at 40–50%.

As is to be expected, the 'OoB + both aux – unweighted + swsn' run is right most times at high levels of lexical overlap. When it is wrong it produced false positives, i.e., it predicts semantic similarity too often, which is easily explained by the high similarity between the sentences. An interesting glitch is the tiny FP bar at the far right of the figure (90%–100% overlap), which denotes 3 cases of high lexical overlap, where the annotators judged the test sentences not to be semantically similar, while our method for predicting short text semantic similarity did. Indeed, the differences are rather subtle, as illustrated by this example pair:

> *Air Canada, the largest airline in Canada and No. 11 in the world, has been under court protection from creditors since April 1*

and

> *The No. 11 airline in the world, Air Canada has been under court protection from creditors since April 1.*

There is a peak in the middle for the FN bars, at 50–60% overlap, where our method cannot make up for the relatively low level of lexical overlap, and mistakenly predicts

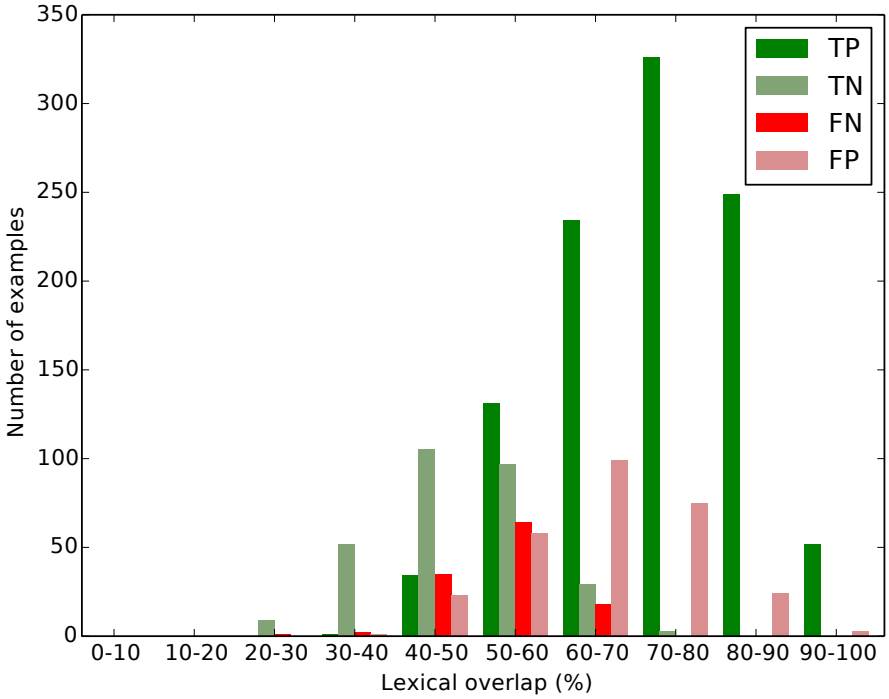


Figure 5.3: Results for the 'OoB + both aux – unweighted + swsn' run, grouped by percentage of lexical overlap between test sentences in the MSR Paraphrase Corpus.

semantic dissimilarity. Interestingly, though, the bars show that our algorithm, even for these difficult cases, still makes the correct prediction in the majority of cases. An additional noteworthy observation is that even when the majority of words in the sentences that make up the test pairs are different, at 40–50% overlap level, our algorithm still produces true positives, next to true negatives. This clearly shows the benefit of semantic matching over lexical matching. More importantly, it shows a meaningful distinction can be made by the algorithm, even for these non-trivial cases.

The final important observation from Figure 5.3 is that our method — the 'OoB both aux – unweighted + swsn' run — is right in the majority of cases across all levels of lexical overlap.

5.5 Additional experiments

As discussed in §5.3, we perform additional experiments on the SemEval 2012 STS and SemEval 2013 STS sets.[9] Tables 5.3 and 5.4 show the results of our experiments for different combinations of features. For easy reference, the results of the best performing team per year are displayed on the top row.

Both on the SemEval 2012 STS and SemEval 2013 sets, cross validation for the experiments with only OoB vectors leads to settings of C $= 10^6$ and gamma values between 10^{-4} or 10^{-6} when just dst-bns features are used (we used folds of size 50

[9]Note that, as mentioned in §5.3.6 the MSRpar subset in the SemEval 2012 STS data is different from the full MSR Paraphrase Corpus used in the previous section.

Table 5.3: Results on SemEval 2012 STS test sets. Last column lists the score in terms of weighted averaged Pearson correlation.

		MSRpar	MSRvid	SMT-europarl	OnWN	SMT-news	*score*
No. 1 SemEval 2012		.683	.874	.528	.664	.494	.677
Our method	Features						
OoB	dst-bns	.643	.818	.424	.621	.485	.627
OoB	dst-bns, SemBM25	.639	.843	.565	.624	.495	.655
OoB + both aux	dst-bns	.646	.820	.536	.632	.490	.648
OoB + both aux	dst-bns, SemBM25	.641	.832	.522	.636	.495	.650

Table 5.4: Results on SemEval 2013 STS test sets. Last column lists the score in terms of weighted averaged Pearson correlation.

		FNWN	headlines	OnWN	SMT	*score*
No. 1 SemEval 2013		.582	.764	.752	.380	.618
Our method	features					
OoB	dst-bns	.332	.613	.682	.342	.516
OoB	dst-bns, SemBM25	.421	.576	.688	.356	.518
OoB + both aux	dst-bns	.448	.712	.683	.403	.580
OoB + both aux	dst-bns, SemBM25	.479	.733	.733	.401	.601

for the more sizeable SemEval 2013 sets). When the SemBM25 features are added, identical values for the hyperparameters turn out to be optimal in cross validation, though gamma should never be as high as 10^{-4} with this number of features.

As can be observed from Table 5.3 the results with only OoB features yield consistent performance. The most important observation is that our generic method, which used only semantic features obtained in an unsupervised way, can outperform the best team of SemEval 2012 on several subsets (SMTeuroparl and SMTnews). Adding SemBM25 features consistently increases performance, but in one case on the MSRpar set where the cross validation picked a setting with somewhat inferior performance. We see that for the SemEval 2012 sets, which have limited training data available, adding both sets of auxiliary vectors does not yield substantial improvements and can even lead to worse performance. This is a clear sign of overfitting where there are too many features for too few training examples.

In Table 5.4, we observe that our method can outperform the best SemEval participant of that year on a subset (SMT). We also see, however, that the mismatch between

training and test material in the SemEval 2013 sets can lead to suboptimal performance. When the auxiliary vectors are added, results are more consistent and the overall performance in terms of weighted average Pearson correlation is close to the score of best performing team.

The fact that our semi-supervised generic method can yield performance on par with the best performing SemEval participants is particularly noteworthy as among the top runs a large number of external sources of structured semantic knowledge is used, such as WordNet, Wikipedia and Wiktionary, next to linguistic tools such as dependency parsers, lemmatizers, POS taggers, SMT systems, stop word lists and NER tools, and handcrafted rules that that normalize currency values and that deal with negations, compound noun phrases and numbers.

5.6 Conclusions

In this chapter, the research question under consideration was:

RQ2 How can pre-trained word embeddings be used to calculate similarity between short texts, without relying on linguistic structure?

We described a generic and flexible method for semantic matching of short texts, which leverages word embeddings of different dimensionality, obtained by different algorithms and from different sources. The method makes no use of external sources of structured semantic knowledge nor of linguistic tools, such as parsers. Instead, it uses a word alignment method, and a saliency-weighted semantic graph, to go from word-level to short-text-level semantics. We computed features from the word alignment method and from the means of word embeddings, to train a final classifier that predicts a semantic similarity score.

We demonstrated on a large publicly available evaluation set that our generic, semantics-only method of computing semantic similarity between short texts outperforms all baseline approaches working under the same conditions, and that it exceeds all approaches using external sources of structured semantic knowledge that, to our knowledge, were evaluated in this dataset.

An important implication of our results is that distributional semantics has come to a level where it can be employed by itself in a generic approach for producing features that can be used to yield state-of-the-art performance on the short text similarity task, even if no manually tuned features are added that optimize for a specific test set or domain. Furthermore, the word embeddings, when employed as we proposed, substitute external semantic knowledge and make human "feature engineering" unnecessary. As our method does not depend on NLP tools, it can be applied to domains and languages for which these are sparse.

In this chapter we proposed a way of going from word-level semantics to short-text level semantics using pre-trained word embeddings. In the next chapter, we propose a method for training word embeddings from scratch, optimizing them explicitly for the task of semantically representing short texts.

6

Siamese CBOW

Optimizing Word Embeddings for Sentence Representations

6.1 Introduction

Word embeddings have proven to be beneficial in a variety of tasks in NLP such as machine translation [172], parsing [30], semantic search [128, 151], and tracking the meaning of words and concepts over time [83, 89]. It is not evident, however, how word embeddings should be combined to represent larger pieces of text, like sentences, paragraphs or documents. Surprisingly, simply averaging word embeddings of all words in a text has proven to be a strong baseline or feature across a multitude of tasks [43, 49, 79, 169].

Word embeddings, however, are not optimized specifically for representing sentences. In this chapter we present a model for obtaining word embeddings that are tailored specifically for the task of averaging them. We do this by directly including a comparison of sentence embeddings — the averaged embeddings of the words they contain — in the cost function of a network that trains word embeddings.

Word embeddings are typically trained in a fast and scalable way from unlabeled training data. As the training data is unlabeled, word embeddings are usually not task-specific. Rather, word embeddings trained on a large training corpus, like the ones from [35, 119] are employed across different tasks [70, 79, 140]. These two qualities — (i) being trainable from large quantities of unlabeled data in a reasonable amount of time, and (ii) robust performance across different tasks — are highly desirable and allow word embeddings to be used in many large-scale applications. In this work we aim to optimize word embeddings for sentence representations in the same manner. We want to produce general purpose sentence embeddings that should score robustly across multiple test sets, and we want to leverage large amounts of unlabeled training material.

In the word2vec algorithm, Mikolov et al. [118] construe a supervised training criterion for obtaining word embeddings from unsupervised data, by predicting, for every word, its surrounding words. We apply this strategy at the sentence level, where we aim to predict a sentence from its adjacent sentences [67, 92]. This allows us to use

unlabeled training data, which is easy to obtain; the only restriction being that sentence boundaries need to be available and that the order between sentences should be known.

The research question we address is:

RQ3 Is it beneficial for word embeddings to be optimized for the task of being averaged to represent short texts?

Specifically, we want to investigate whether directly optimizing word embeddings for the task of being averaged to produce sentence embeddings leads to word embeddings that are better suited for this task than word2vec does. Therefore, we test the embeddings in an unsupervised learning scenario. We use 20 evaluation sets that stem from a wide variety of sources (newswire, video descriptions, dictionary descriptions, microblog posts). Furthermore, we analyze the time complexity of our method and compare it to our baselines methods.
We first present our model for obtaining word embeddings optimized for sentence embeddings in §6.2. The experimental setup and outcomes are detailed in §6.3 and §6.4, respectively.

6.2 Siamese CBOW

We present the *Siamese Continuous Bag of Words* (CBOW) model, a neural network for efficient estimation of high-quality sentence embeddings. Quality should manifest itself in embeddings of semantically close sentences being similar to one another, and embeddings of semantically different sentences being dissimilar. An efficient and surprisingly successful way of computing a sentence embedding is to average the embeddings of its constituent words. Recent work uses pre-trained word embeddings (such as word2vec and GloVe) for this task, which are not optimized for sentence representations. Following these approaches, we compute sentence embeddings by averaging word embeddings, but we optimize word embeddings directly for the purpose of being averaged.

6.2.1 Training objective

We construct a supervised training criterion by having our network predict sentences occurring next to each other in the training data. Specifically, for a pair of sentences (s_i, s_j), we define a probability $p(s_i, s_j)$ that reflects how likely it is for the sentences to be adjacent to one another in the training data. We compute the probability $p(s_i, s_j)$ using a softmax function:

$$p_\theta(s_i, s_j) = \frac{e^{\cos(\mathbf{s_i^\theta}, \mathbf{s_j^\theta})}}{\sum_{s_{i'} \in S} e^{\cos(\mathbf{s_i^\theta}, \mathbf{s_{i'}^\theta})}}, \tag{6.1}$$

where $\mathbf{s}_x^\theta$ denotes the embedding of sentence s_x, based on the model parameters θ. In theory, the summation in the denominator of Equation 6.1 should range over all possible sentences S, which is not feasible in practice. Therefore, we replace set S with the union of two sets: set S^+, containing sentences that occur next to sentence

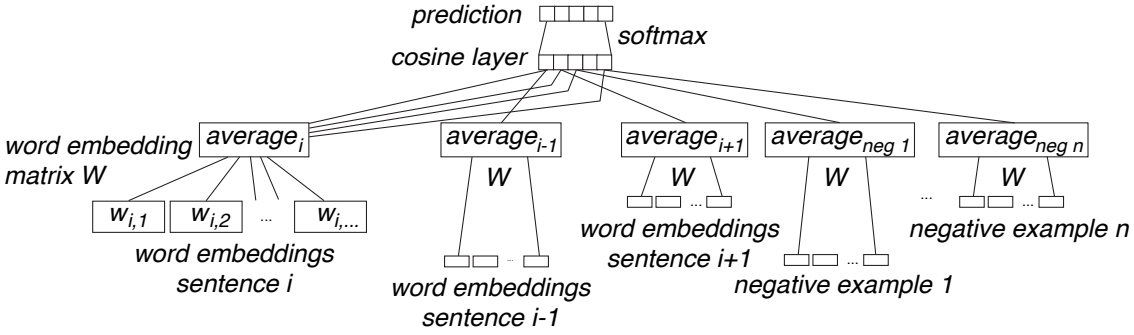


Figure 6.1: Siamese CBOW network architecture. (Input projection layer omitted.)

s_i in the training data, and set S^-, a set of n randomly chosen sentences that are not observed next to the sentence s_i in the training data. The loss function of the network is categorical cross entropy, mentioned earlier in Chapter 2, Equation 2.2. Applied to current setting, we have:

$$L = - \sum_{s_j \in \{S^+ \cup S^-\}} p(s_i, s_j) \log p_\theta(s_i, s_j),$$

where $p(\cdot)$ is the target probability the network should produce, and $p_\theta(\cdot)$ is the prediction it estimates based on parameters θ, using Equation 6.1. The target distribution simply is:

$$p(s_i, s_j) = \begin{cases} \frac{1}{|S^+|}, & \text{if } s_j \in S^+ \\ 0, & \text{if } s_j \in S^-. \end{cases}$$

I.e., if there are 2 positive examples (the sentences preceding and following the input sentence) and 2 negative examples, the target distribution is (0.5, 0.5, 0, 0).

6.2.2 Network architecture

Figure 6.1 shows the architecture of the proposed Siamese CBOW network. The input is a projection layer that selects embeddings from a word embedding matrix W (that is shared across inputs) for a given input sentence. The word embeddings are averaged in the next layer, which yields a sentence representation with the same dimensionality as the input word embeddings (the boxes labeled *average*$_i$ in Figure 6.1). The cosine similarities between the sentence representation for *sentence*$_i$ and the other sentences are calculated in the penultimate layer and a softmax is applied in the last layer to produce the final probability distribution.

6.2.3 Training

The weights in the word embedding matrix are the only trainable parameters in the Siamese CBOW network. They are updated using stochastic gradient descent. The initial learning rate is monotonically decreased proportionally to the number of training batches.

6.3 Experimental setup

To answer RQ3, we test the efficacy of our siamese network for producing sentence embeddings we use multiple test sets. We use Siamese CBOW to learn word embeddings from an unlabeled corpus. For every sentence pair in the test sets, we compute two sentence representations by averaging the word embeddings of each sentence. Words that are missing from the vocabulary and, hence, have no word embedding, are omitted. The cosine similarity between the two sentence vectors is produced as a final semantic similarity score.

As we want a clean way to directly evaluate the embeddings on multiple sets we train our model and the models we compare with on exactly the same training data. We do not compute extra features, perform extra preprocessing steps or incorporate the embeddings in supervised training schemes. Additional steps like these likely improve evaluation scores, but they would obscure our main evaluation purpose in this chapter, which is to directly test the embeddings.

6.3.1 Data

We use the Toronto Book Corpus[1] [171] to train word embeddings. The corpus contains 74,004,228 already pre-processed sentences in total, which contain 1,057,070,918 tokens, originating from 7,087 unique books. In our experiments, we only consider tokens appearing 5 times or more, which leads to a vocabulary of 315,643 words.

6.3.2 Baselines

We employ two baselines for producing sentence embeddings in our experiments. We obtain similarity scores between sentence pairs from the baselines in the same way as the ones produced by Siamese CBOW, i.e., we calculate the cosine similarity between the sentence embeddings they produce.

Word2vec We average word embeddings trained with word2vec.[2] We use both architectures, Skip-gram and CBOW, and apply default settings: minimum word frequency 5, word embedding size 300, context window 5, sample threshold 10^{-5}, no hierarchical softmax, 5 negative examples.

Skip-thought As a second baseline we use the sentence representations produced by the skip-thought architecture [92].[3] Skip-thought is a recently proposed method that learns sentence representations in a different way from ours, by using recurrent neural networks. This allows it to take word order into account. As it trains sentence embeddings from unlabeled data, like we do, it is a natural baseline to consider.

[1]The corpus can be downloaded from `http://www.cs.toronto.edu/~mbweb/`.

[2]The code is available from `https://code.google.com/archive/p/word2vec/`.

[3]The code and the trained models can be downloaded from `https://github.com/ryankiros/skip-thoughts/`.

Both methods are trained on the Toronto Book Corpus, the same corpus used to train Siamese CBOW. We should note that as we use skip-thought vectors as trained by Kiros et al. [92], skip-thought has an advantage over both word2vec and Siamese CBOW as the vocabulary used for encoding sentences contains 930,913 words, three times the size of the vocabulary we use.

6.3.3 Evaluation

We use 20 SemEval datasets from the SemEval semantic textual similarity task in 2012, 2013, 2014 and 2015 [2–5], which consist of sentence pairs from a wide array of sources (e.g., newswire, tweets, video descriptions) that have been manually annotated by multiple human assessors on a 5 point scale (1: semantically unrelated, 5: semantically similar). In the ground truth, the final similarity score for every sentence pair is the mean of the annotator judgements, and as such can be a floating point number like 2.685.

The evaluation metric used by SemEval, and hence by us, is Pearson's r. As Spearman's r is often reported as well, we do so too.

Statistical significance To see whether Siamese CBOW yields significantly different scores for the same input sentence pairs from word2vec CBOW — the method it is theoretically most similar to — we compute Wilcoxon signed-rank test statistics between all runs on all evaluation sets. Runs are considered statistically significantly different for p-values < 0.0001.

6.3.4 Network

To comply with results reported in other research [100, 119] we fix the embedding size to 300 and only consider words appearing 5 times or more in the training corpus. We use 2 negative examples (see §6.4.2 for an analysis of different settings). The embeddings are initialized randomly, by drawing from a normal distribution with $\mu = 0.0$ and $\sigma = 0.01$. The batch size is 100. The initial learning rate α is 0.0001, which we obtain by observing the loss on the training data. Training consists of one epoch.

We use Theano [146] to implement our network.[4] We ran our experiments on GPUs in the DAS5 cluster [11].

6.4 Results

In this section we present the results of our experiments, and analyze the stability of Siamese CBOW with respect to its (hyper)parameters.

Table 6.1: Results on SemEval datasets in terms of Pearson's r (Spearman's r). Highest scores, in terms of Pearson's r, are displayed in bold. Siamese CBOW runs statistically significantly different from the word2vec CBOW baseline runs are marked with a †. See §6.3.3 for a discussion of the statistical test used.

Dataset	w2v skipgram	w2v CBOW	skip-thought	Siamese CBOW
2012				
MSRpar	0.3740 (0.3991)	0.3419 (0.3521)	0.0560 (0.0843)	**0.4379**† (0.4311)
MSRvid	0.5213 (0.5519)	0.5099 (0.5450)	**0.5807** (0.5829)	0.4522† (0.4759)
OnWN	0.6040 (0.6476)	0.6320 (0.6440)	0.6045 (0.6431)	**0.6444**† (0.6475)
SMTeuroparl	0.3071 (0.5238)	0.3976 (0.5310)	0.4203 (0.4999)	**0.4503**† (0.5449)
SMTnews	**0.4487** (0.3617)	0.4462 (0.3901)	0.3911 (0.3628)	0.3902† (0.4153)
2013				
FNWN	**0.3480** (0.3401)	0.2736 (0.2867)	0.3124 (0.3511)	0.2322† (0.2235)
OnWN	0.4745 (0.5509)	**0.5165** (0.6008)	0.2418 (0.2766)	0.4985† (0.5227)
SMT	0.1838 (0.2843)	0.2494 (0.2919)	**0.3378** (0.3498)	0.3312† (0.3356)
headlines	0.5935 (0.6044)	0.5730 (0.5766)	0.3861 (0.3909)	**0.6534**† (0.6516)
2014				
OnWN	0.5848 (0.6676)	0.6068 (0.6887)	0.4682 (0.5161)	**0.6073**† (0.6554)
deft-forum	0.3193 (0.3810)	0.3339 (0.3507)	0.3736 (0.3737)	**0.4082**† (0.4188)
deft-news	0.5906 (0.5678)	0.5737 (0.5577)	0.4617 (0.4762)	**0.5913**† (0.5754)
headlines	0.5790 (0.5544)	0.5455 (0.5095)	0.4031 (0.3910)	**0.6364**† (0.6260)
images	0.5131 (0.5288)	0.5056 (0.5213)	0.4257 (0.4233)	**0.6497**† (0.6484)
tweet-news	0.6336 (0.6544)	0.6897 (0.6615)	0.5138 (0.5297)	**0.7315**† (0.7128)
2015				
answ-forums	0.1892 (0.1463)	0.1767 (0.1294)	**0.2784** (0.1909)	0.2181 (0.1469)
answ-students	0.3233 (0.2654)	0.3344 (0.2742)	0.2661 (0.2068)	**0.3671**† (0.2824)
belief	0.2435 (0.2635)	0.3277 (0.3280)	0.4584 (0.3368)	**0.4769** (0.3184)
headlines	0.1875 (0.0754)	0.1806 (0.0765)	0.1248 (0.0464)	**0.2151**† (0.0846)
images	0.2454 (0.1611)	0.2292 (0.1438)	0.2100 (0.1220)	**0.2560**† (0.1467)

6.4.1 Main experiments

In Table 6.1, the results of Siamese CBOW on 20 SemEval datasets are displayed, together with the results of the baseline systems. As we can see from the table, Siamese CBOW outperforms the baselines in the majority of cases (14 out of 20). The very low scores of skip-thought on MSRpar appear to be a glitch, which we will ignore.

It is interesting to see that for the set with the highest average sentence length (2013 SMT, with 24.7 words per sentence on average) Siamese CBOW is very close to skip-thought, the best performing baseline. In terms of lexical term overlap, unsurprisingly, all methods have trouble with the sets with little overlap (2013 FNWN, 2015 answers-forums, which both have 7% lexical overlap). It is interesting to see, however, that for

[4]The code for Siamese CBOW is available under an open-source license at `https://bitbucket.org/TomKenter/siamese-cbow`.

Table 6.2: Results on SemEval 2014 datasets in terms of Pearson's r (Spearman's r). Highest scores (in Pearson's r) are displayed in bold. FastSent results are reprinted from [67] where they are reported in two-digit precision.

Dataset	FastSent	Siamese CBOW
OnWN	**0.74** (0.70)	0.6073 (0.6554)
deft-forum	**0.41** (0.36)	0.4082 (0.4188)
deft-news	0.58 (0.59)	**0.5913** (0.5754)
headlines	0.57 (0.59)	**0.6364** (0.6260)
images	**0.74** (0.78)	0.6497 (0.6484)
tweet-news	0.63 (0.66)	**0.7315** (0.7128)

the next two sets (2015 belief and 2012 MSRpar, 11% and 14% overlap respectively) Siamese CBOW manages to get the best performance. The highest performance on all sets is 0.7315 Pearson's r of Siamese CBOW on the 2014 tweet-news set. This figure is not very far from the best performing SemEval run that year which has 0.792 Pearson's r. This is remarkable as Siamese CBOW is completely unsupervised, while the NTNU system which scored best on this set [111] was optimized using multiple training sets.

In recent work, Hill et al. [67] present FastSent, a model similar to ours (see §3.2.2 for a more elaborate discussion); results are not reported for all evaluation sets we use, and hence, we compare the results of FastSent and Siamese CBOW separately, in Table 6.2.

FastSent and Siamese CBOW each outperform the other on half of the evaluation sets, which clearly suggests that the differences between the two methods are complementary.[5]

6.4.2 Analysis

Next, we investigate the stability of Siamese CBOW with respect to its hyper-parameters. In particular, we look into stability across iterations, different numbers of negative examples, and the dimensionality of the embeddings. Other parameter settings are set as reported in §6.3.4.

Performance across iterations

Ideally, the optimization criterion of a learning algorithm ranges over the full domain of its loss function. As discussed in §6.2, our loss function only observes a sample. As such, convergence is not guaranteed. Regardless, an ideal learning system should not fluctuate in terms of performance relative to the amount of training data it observes, provided this amount is substantial: as training proceeds the performance should stabilize.

To see whether the performance of Siamese CBOW fluctuates during training we monitor it during 5 epochs; at every 10,000,000 examples, and at the end of every epoch.

[5]The comparison is to be interpreted with caution as it is not evident what vocabulary was used for the experiments in [67]; hence, the differences observed here might simply be due to differences in vocabulary coverage.

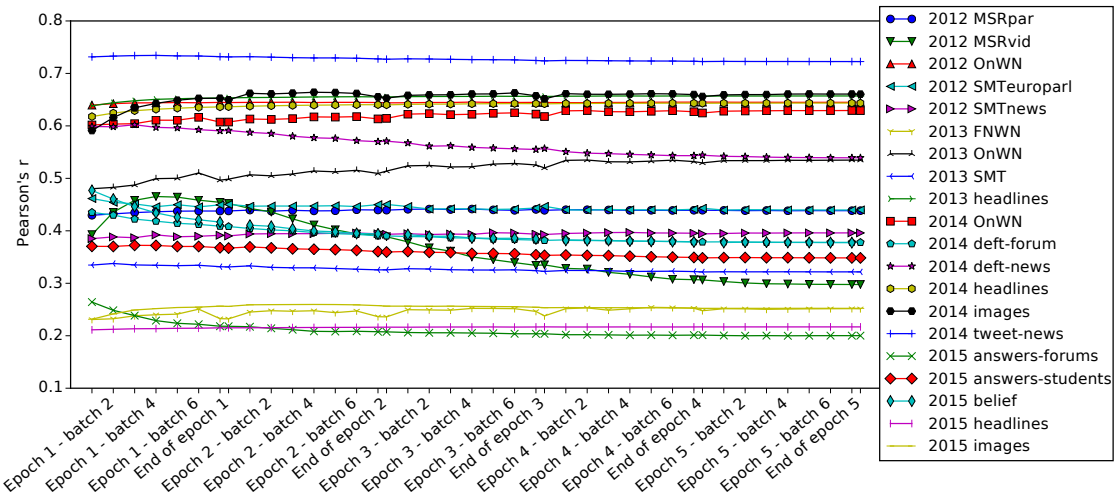


Figure 6.2: Performance of Siamese CBOW across 5 iterations.

Figure 6.2 displays the results for all 20 datasets. We observe that on the majority of datasets the performance shows very little variation. There are three exceptions. The performance on the 2014 deft-news dataset steadily decreases while the performance on 2013 OnWN steadily increases, though both seem to stabilize at the end of epoch 5. The most notable exception, however, is 2012 MSRvid, where the score, after an initial increase, drops consistently. This effect might be explained by the fact that this evaluation set primarily consists of very short sentences — it has the lowest average sentence length of all set: 6.63 with a standard deviation of 1.812. Therefore, a 300-dimensional representation appears too large for this dataset; this hypothesis is supported by the fact that 200-dimensional embeddings work slightly better for this dataset (see Figure 6.4).

Number of negative examples

In Figure 6.3, the results of Siamese CBOW in terms of Pearson's r are plotted for different numbers of negative examples. We observe that on most sets, the number of negative examples has limited effect on the performance of Siamese CBOW. Choosing a higher number, like 10, occasionally leads to slightly better performance, e.g., on the 2013 FNWN set. However, a small number like 1 or 2 typically suffices, and is sometimes markedly better, e.g., in the case of the 2015 belief set. As a high number of negative examples comes at a substantial computational cost, we conclude from the findings presented here that, although Siamese CBOW is robust against different settings of this parameter, setting the number of negative examples to 1 or 2 should be the default choice.

Number of dimensions

Figure 6.4 plots the results of Siamese CBOW for different numbers of vector dimensions. We observe from the figure that for some sets (most notably 2014 deft-forum, 2015 answ-forums and 2015 belief) increasing the number of embedding dimensions

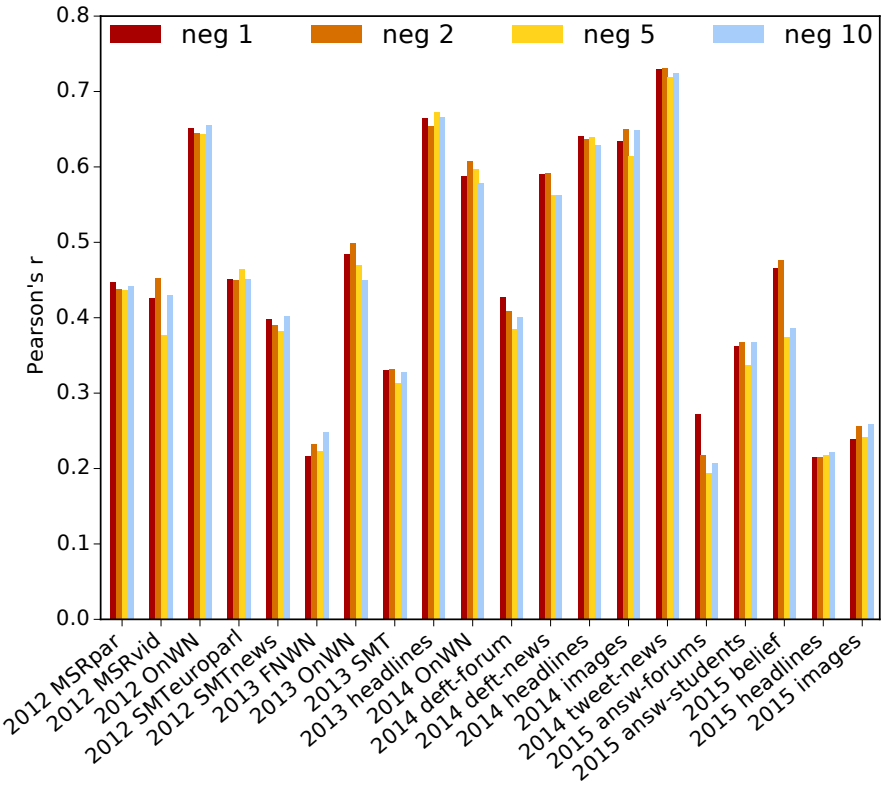


Figure 6.3: Performance of Siamese CBOW with different numbers of negative examples.

consistently yields higher performance. A dimensionality that is too low (50 or 100) invariably leads to inferior results. As, similar to a higher number of negative examples, a higher embedding dimension leads to higher computational costs, we conclude from these findings that a moderate number of dimensions (200 or 300) is to be preferred.

6.4.3 Time complexity

For learning systems, time complexity comes into play in the training phase and in the prediction phase. For an end system employing sentence embeddings, the complexity at prediction time is the most crucial factor, which is why we omit an analysis of training complexity. We focus on comparing the time complexity for generating sentence embeddings for Siamese CBOW, and compare it to the baselines we use.

The complexity of all algorithms we consider is $\mathcal{O}(n)$, i.e., linear in the number of input terms. As in practice the number of arithmetic operations is the critical factor in determining computing time, we will now focus on these.

Both word2vec and the Siamese CBOW compute embeddings of a text $T = t_1, \ldots, t_{|T|}$ by averaging the term embeddings. This requires $|T| - 1$ vector additions, and 1 multiplication by a scalar value (namely, $1/|T|$). The skip-thought model is a recurrent neural network with GRU cells, which computes a set of equations for every

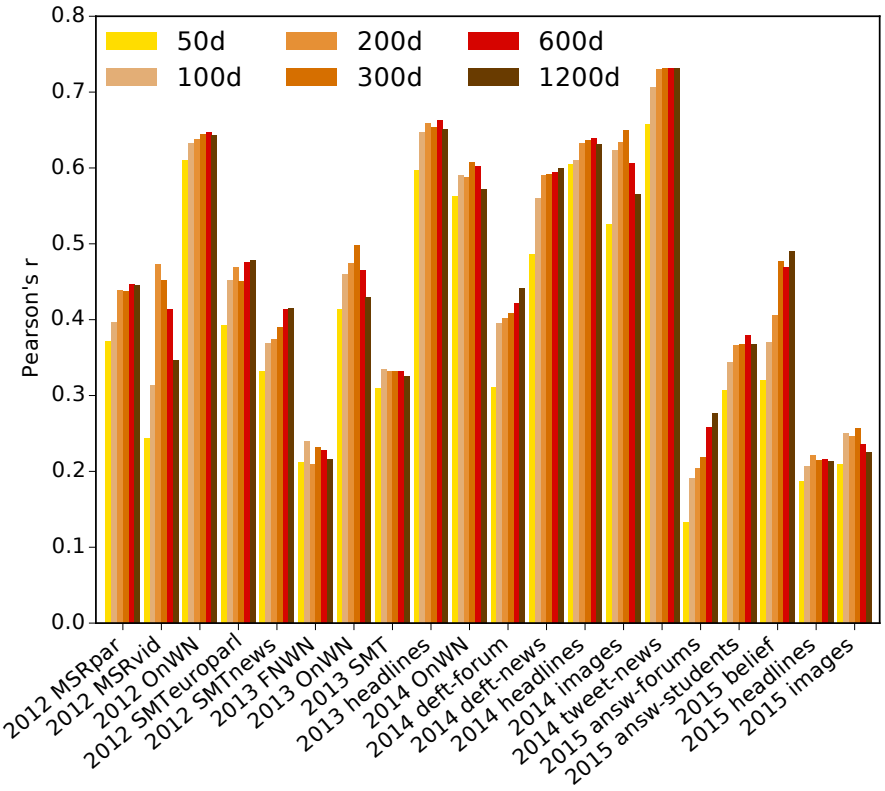


Figure 6.4: Performance of Siamese CBOW across number of embedding dimensions.

Table 6.3: Time spent per method on all 20 SemEval datasets, 17,608 sentence pairs, and the average time spent on a single sentence pair (time in seconds unless indicated otherwise).

	20 sets	1 pair
Siamese CBOW (300d)	7.7	0.0004
word2vec (300d)	7.0	0.0004
skip-thought (1200d)	98,804.0	5.6

term t in T, which we reprint for reference [92]:

$$\begin{aligned}
\mathbf{r}^t &= \sigma(\mathbf{W}_r\mathbf{x}^t + \mathbf{U}_r\mathbf{h}^{t-1}) \\
\mathbf{z}^t &= \sigma(\mathbf{W}_z\mathbf{x}^t + \mathbf{U}_z\mathbf{h}^{t-1}) \\
\overline{\mathbf{h}}^t &= \tanh(\mathbf{W}\mathbf{x}^t + \mathbf{U}(\mathbf{r}^t \odot \mathbf{h}^{t-1})) \\
\mathbf{h}^t &= (1 - \mathbf{z}^t) \odot \mathbf{h}^{t-1} + \mathbf{z}^t \odot \overline{\mathbf{h}}^t
\end{aligned}$$

As we can see from the formulas, there are $5|T|$ vector additions (+/-), $4|T|$ element-wise multiplications by a vector, $3|T|$ element-wise operations and $6|T|$ matrix multiplications, of which the latter, the matrix multiplications, are most expensive.

This considerable difference in numbers of arithmetic operations is also observed in practice. We run tests on a single CPU, using identical code for extracting sentences from the evaluation sets, for every method. The sentence pairs are presented one by one

to the models. We disregard the time it takes to load models. Speedups might of course be gained for all methods by presenting the sentences in batches to the models, by computing sentence representations in parallel and by running code on a GPU. However, as we are interested in the differences between the systems, we run the most simple and straightforward scenario. Table 6.3 lists the number of seconds each method takes to generate and compare sentence embeddings for an input sentence pair. The difference between word2vec and Siamese CBOW is because of a different implementation of word lookup.

We conclude from the observations presented here, together with the results in §6.4.1, that in a setting where speed at prediction time is pivotal, simple averaging methods like word2vec or Siamese CBOW are to be preferred over more involved methods like skip-thought.

6.4.4 Qualitative analysis

As Siamese CBOW directly averages word embeddings for sentences, we expect it to learn that words with little semantic impact have a low vector norm. Indeed, we find that the 10 words with lowest vector norm are *to*, *of*, *and*, *the*, *a*, *in*, *that*, *with*, *on*, and *as*. At the other side of the spectrum we find many personal pronouns: *had*, *they*, *we*, *me*, *my*, *he*, *her*, *you*, *she*, *I*, which is natural given that the corpus on which we train consists of fiction, which typically contains dialogues.

It is interesting to see what the differences in related words are between Siamese CBOW and word2vec when trained on the same corpus. For example, for a cosine similarity > 0.6, the words related to *her* in word2vec space are *she*, *his*, *my* and *hers*. For Siamese CBOW, the only closely related word is *she*. Similarly, for the word *me*, word2vec finds *him* as most closely related word, while Siamese CBOW comes up with *I* and *my*. It seems from these few examples that Siamese CBOW learns to be very strict in choosing which words to relate to each other.

From the results presented in this section we conclude that optimizing word embeddings for the task of being averaged across sentences with Siamese CBOW leads to embeddings that are effective in a large variety of settings. Furthermore, Siamese CBOW is robust to different parameter settings and its performance is stable across iterations. Lastly, we show that Siamese CBOW is fast and efficient in computing sentence embeddings at prediction time.

6.5 Conclusions

In this chapter, we presented Siamese CBOW, a neural network architecture that efficiently learns word embeddings optimized for producing sentence representations. The model is trained using only unlabeled text data. It predicts, from an input sentence representation, the preceding and following sentence. In particular, our aim was to answer the following research question:

RQ3 Is it beneficial for word embeddings to be optimized for the task of being averaged to represent short texts?

We evaluated the model on 20 test sets and show that in a majority of cases, 14 out of 20, Siamese CBOW outperforms a word2vec baseline and a baseline based on the recently proposed skip-thought architecture. From these results we conclude that optimizing word embeddings for the task of being averaged across sentences with Siamese CBOW leads to embeddings that are effective in a large variety of settings. Furthermore, Siamese CBOW is robust to different parameter settings and its performance is stable across iterations. Lastly, we show that Siamese CBOW is fast and efficient in computing sentence embeddings at prediction time.

Our results show that, although averaging generic word embeddings, not trained for a particular task, to represent short tasks can yield strong baseline, optimizing the embeddings for this task can be beneficial. A similar phenomenon may occur across other tasks, such as optimizing embeddings to represent entities or user queries in an IR system.

This chapter concludes the part of this thesis about text understanding at short-text level. Next, we turn to presenting our research concerning natural language understanding at the level of documents.

Part III

Documents

7

Attentive Memory Networks for Natural Language Understanding

7.1 Introduction

Recent advances in conversational systems [127, 135] have changed the search paradigm. In a classic setting, a search engine answers a query based on an index, possibly enriching it with information from an external knowledge base [156]. Additionally, previous interactions in the same session can be leveraged [41]. In addition to these sources, in natural language conversations, information contained in previous utterances can be referred to, even implicitly. Suppose a conversational system has to answer the query *Where are my keys?* based on a previous statement *I was home before I went to work, which is where I found out I didn't have my keys with me*. The statement conveys a lot of information, including the likely possibility that the keys are still at the speaker's house. As is clear from this example, indices or external knowledge bases are of no avail in this setting. It is crucial for a conversational system to maintain an internal state, representing the dialogue with the user so far. To address this issue, substantial work has been done in goal-oriented dialogues, tailored to specific settings such as restaurant reservations [23] and the tourist domain [87]. We argue that a generic conversational agent should be able to maintain a dialogue state without being constrained to a particular task with predetermined slots to be filled. The time has come for the IR community to address the task of machine reading for conversational search [127].

As an important step towards generic conversational IR [93], we frame the task of conversational search as a general machine reading task [64, 65], where a number of statements is provided to an automated agent that answers questions about it. This scenario is different from the traditional question answering setting, in which questions are typically factoid in nature, and answers are based on background knowledge or external sources of knowledge. In the machine reading task, much as in a natural conversation, a number of statements is provided, and the conversational agent should be able to answer questions based on its understanding of these statements alone. In [65], for example, a single Wikipedia page is provided to a machine algorithm which has to answer questions about it. In [159] the machine reads stories about persons and objects and has to keep track of their whereabouts.

Memory networks have proven to be an effective architecture in machine reading tasks [144, 158]. Their key component is a memory module in which the model stores intermediate representations of input, that can be seen as multiple views on the input so far, from which a final output is computed. Speed is an important constraint in the context of conversational agents, since long pauses between turns hamper the naturalness of a conversation. The research question we aim to answer in this chapter is:

RQ4 Can an efficient memory network be designed using RNNs with attention mechanism only, without loosing performance?

As we strive for an efficient architecture, we propose to use a hierarchical input encoder. Input can be large, hundreds of words, and we hypothesize that first processing the input to get a smaller set of higher-level input representations can benefit a network in two ways: (1) the higher-level representations provide a distilled representation of the input; (2) as there are fewer higher-level representations it should be (computationally) easier for the network to focus on the relevant parts of the input. In short, in this chapter we present the *Attentive Memory Network* (AMN), an end-to-end trainable memory network, with hierarchical input encoder. To test its general applicability we use 20 machine reading datasets specifically designed to highlight different aspects of natural language understanding.

7.2 Attentive memory networks

AMNs, like traditional sequence-to-sequence networks, are composed of recurrent neural networks. These are discussed in detail in §2.2.

7.2.1 Attentive memory network architecture

We now present the Attentive Memory Network (AMN) architecture. The key part of any memory network is a memory module, which is a recurrent network itself. It stores memories by attending over the input document, conditioned on the question. As can be seen from Equation 2.5, the computational complexity of the attention mechanism is primarily dependent on the size of $\mathbf{H}^{att}$, the states to attend over. To keep this matrix small, a hierarchical approach is taken, where the input is first read by a word-level document encoder, which reads word embeddings — also trained by the model — per sentence to compute sentence representations. A sentence-level encoder iterates over these sentence embeddings to get a final document encoding. The memory module only has access to the sentence embeddings produced by the sentence-level encoder. For example, if the input consists of 20 sentences of 12 words each, the memory module of the AMN attends over 20 sentence representations, rather than over 240 representations, had a non-hierarchical word-level approach been taken.

Figure 7.1 shows a graphical overview of the network layout. There are two input encoders, a question encoder and a word-level document encoder. The memory module, the green block in Figure 7.1, attends over the sentence embeddings to extract relevant parts of the input, conditioned on the question. Lastly, the answer decoder attends

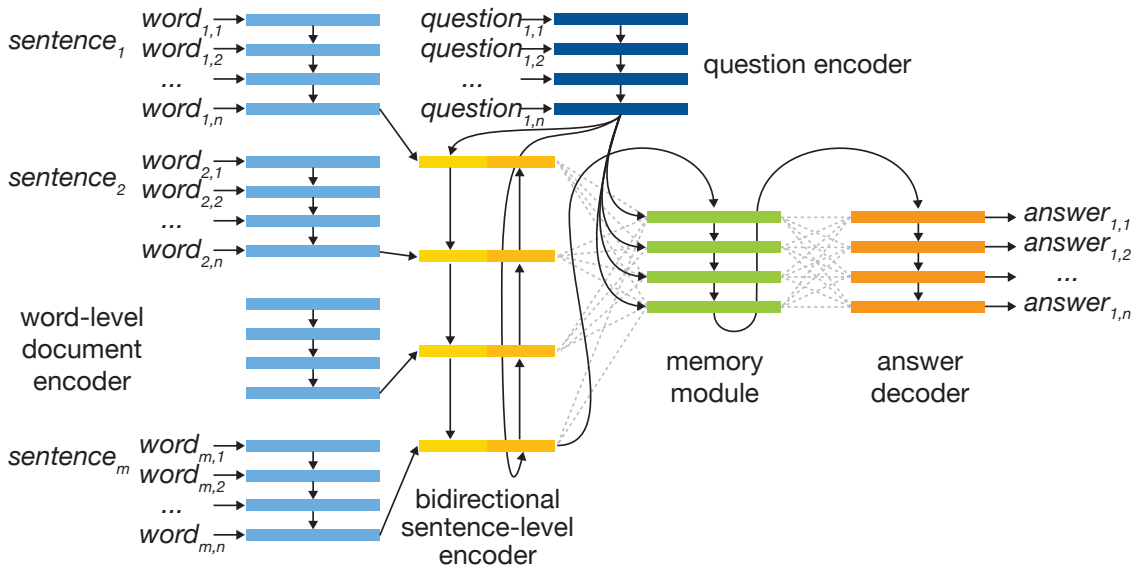


Figure 7.1: Attentive Memory Network. Connected blocks sharing color represent RNNs. Attention is depicted by dashed lines.

over the memory states, to produce the final output. Below we discuss each part of the network in detail.

Question encoder

For encoding the question we use a single RNN. For a question $Q \in \{q_1, q_2, \ldots, q_{|Q|}\}$ it produces a final state $\mathbf{h}^{que}_{|Q|}$, a vector of dimension d^{que}, that is used as a distributed representation of the question.

Document encoder

To encode the document we use a hierarchical approach. First, a word-level RNN is used to encode sentences. The word-level encoder is applied for every sentence individually. The unroll length is the maximum sentence length in words. For sentences $S \in \{s_1, s_2, \ldots, s_{|S|}\}$ the word-level encoder yields $\mathbf{H}^{wrd}$, an $|S| \times d^{wrd}$ matrix, as detailed in §2.2.

The sentence representations in $\mathbf{H}^{wrd}$ are read as a sequence by a sentence-level encoder. Following, e.g., [164], we use a bidirectional RNN for the sentence-level encoder, which for $|S|$ sentences and a hidden state size d^{sen} yields $\mathbf{H}^{sen}$, an $|S| \times d^{sen}$ matrix. The final state of the question encoder, $\mathbf{h}^{que}_{|Q|}$, is used as initial value of the hidden states of the sentence-level encoder.

Memory module

The memory module consists of a single recurrent cell that produces $\mathbf{M}$, a matrix of m memory representations of dimension d^{mem}. The i-th memory $\mathbf{m}_i$ is computed

conditioned on the question representation and the sentence representations, analogical to Equation 2.5, as:

$$\mathbf{m}_i = g\left(\mathbf{h}^{que}_{|Q|}, \mathbf{H}^{sen}, \mathbf{m}_{i-1}\right). \tag{7.1}$$

That is, the final representation of the question encoder $\mathbf{h}^{que}_{|Q|}$ is repeatedly provided as input to a recurrent cell, whose hidden state is computed from the memory it produced previously, $\mathbf{m}_{i-1}$, while attending over the hidden states of the sentence-level encoder $\mathbf{H}^{sen}$.

The final representation of the sentence-level document encoder $\mathbf{h}^{sen}_{|S|}$ is used to initialize the hidden state of the memory cell, $\mathbf{m}_0$.

Answer decoder

Finally, the decoder produces an answer as discussed in §2.2. Applying Equation 2.5 to the current setting, where $\mathbf{h}^{dec}_t$ is computed by attending over the memory states, gives us:

$$\mathbf{h}^{dec}_t = g(\mathbf{x}^{dec}_t, \mathbf{M}, \mathbf{h}^{dec}_{t-1}).$$

7.2.2 Efficiency

As can be seen from Equation 7.1, the memory module is a recurrent cell itself. In previous memory networks, the memory module passes over the input multiple times, updating memory after each pass [99, 164]. The key difference in our approach is that AMNs iterate over the input only once, but *attend* over it multiple times. This is more efficient, as the attention mechanism (Equation 2.5) has far less parameters than an LSTM or GRU recurrent cell, which update multiple gates and an internal state at every time step. The attention mechanism calculates a softmax over the input encodings, the number of which in our case is reduced to number of input sentences, rather than words, by the hierarchical encoder.

The AMN needs relatively few iterations to learn. Details per evaluation set are provided in §7.4.2.

7.3 Experimental setup

To the best of our knowledge, there is currently no conversational search data set (consisting of sequences of utterances plus questions about these utterances) on which we could evaluate AMN. Instead we evaluate AMN on a broad collection of more traditional machine reading datasets. Specifically, we evaluate AMN on the 20 datasets provided by the bAbi tasks [159], of which we use the 10k sets, version 1.2. The sets consist of stories, 2 to over 100 sentences in length, and questions about these stories. The 20 sets are designed to highlight different aspects of natural language understanding like counting, deduction, induction and spatial reasoning. As argued by Kumar et al. [99], while showing the ability to solve one of the bAbi tasks is not sufficient to conclude a model would succeed at the same task on real world text data — such as conversational search data — it is a necessary condition.

Every dataset in the bAbi collection comes as a training set of 10,000 examples and a test set of 1,000 examples. We split the 10,000 training examples of each dataset into a training set — the first 9,000 examples — and a validation set — the remaining 1,000 examples — on which we tune the hyperparameters. All text is lowercased.

We use GRU cells [32] for all recurrent cells. To restrict the number of hyperparameters to tune, the same value is used for all embedding sizes, and for the state sizes of all recurrent cells. I.e., for an embedding size e, we have $e = d^{que} = d^{wrd} = d^{sen} = d^{mem}$, which is either 32 or 64. The weights of the question encoder and document word-level encoder are tied. GRU cells can be stacked and we experiment with 1 to 3 level deep encoder, memory, and decoder cells, the depths of which always match (i.e., if, for example, 3-level encoder cells are used, 3-level decoder cells are used). We use a single embedding matrix for the words in the question, document and answer. The number of memories to generate, m, is chosen from $\{1, 2, 3\}$. Dropout is applied at every recurrent cell, the dropout probability being either 0.0 (no dropout), 0.1 or 0.2. We optimize cross entropy loss, cf. Equation 2.2, between actual and predicted answers, using Adam [91] as optimization algorithm and set the initial learning rate to one of $\{0.1, 0.5, 1.0\}$. We measure performance every 1,000 training examples. If the loss does not improve or performance on the validation set decreases for three times in a row, the learning rate is annealed by dividing it by 2. The maximum norm for gradients is either 1 or 5. The batch size is set to 50.

We implemented the AMN in Tensorflow [1]. The implementation is released under an open source license and is available at `https://bitbucket.org/TomKenter/attentive-memory-networks-code`.

7.4 Results and analysis

We present the results of the experiments described in §7.3 and provide an analysis of the results.

7.4.1 Main results

Table 8.4 lists the results of our Attentive Memory Network (AMN) on the 20 bAbi 10k datasets, together with results of previous approaches. Following [159], we consider a dataset solved when the error rate is less than 5%.

As can be seen from the Table 8.4, AMN solves 18 of the 20 datasets. This is particularly noteworthy given the fact that it is a general framework, not catered towards tracking entities (as in [61]). Moreover, the AMN needs an order of magnitude fewer computation steps than previous memory network architectures used for these tasks [99, 164] as it only reads the input once.

There are two tasks the AMN does not solve. The *basic induction* set proves to be hard for the AMN, as it does for most other networks. More interestingly, the *three supporting facts* sets is problematic as well. This dataset has the longest documents, sometimes over 100 sentences long. Analysis of the results, see below for examples, shows that the probability mass of the attention vectors of the memory module is much

Table 7.1: Results in terms of error rate on the bAbi 10k tasks. For comparison, results of previous work are copied from [144, MemN2N], [54, DNC], [164, DMN+], and [61, EntNet].

Dataset	MemN2N	DNC	DMN+	EntNet	AMN
single supporting fact	0.0	0.0	0.0	0.0	0.0
two supporting facts	0.3	0.4	0.3	0.1	4.1
three supporting facts	2.1	1.8	1.1	4.1	29.1
two arg relations	0.0	0.0	0.0	0.0	0.0
three arg relations	0.8	0.8	0.5	0.3	0.7
yes-no questions	0.1	0.0	0.0	0.2	0.2
counting	2.0	0.6	2.4	0.0	3.1
lists sets	0.9	0.3	0.0	0.5	0.3
simple negation	0.3	0.2	0.0	0.1	0.0
indefinite knowledge	0.0	0.2	0.0	0.6	0.1
basic coreference	0.1	0.0	0.0	0.3	0.0
conjunction	0.0	0.0	0.0	0.0	0.0
compound coreference	0.0	0.1	0.0	1.3	0.0
time reasoning	0.1	0.4	0.2	0.0	3.6
basic deduction	0.0	0.0	0.0	0.0	0.0
basic induction	51.8	55.1	45.3	0.2	45.4
positional reasoning	18.6	12.0	4.2	0.5	1.6
size reasoning	5.3	0.8	2.1	0.3	0.9
path finding	2.3	3.9	0.0	2.3	0.3
agents motivations	0.0	0.0	0.0	0.0	0.0
number of tasks solved	18	18	19	20	18

more spread out across sentences then it is in other sets. That is, the network struggles to keep its attention focused.

The results in Table 8.4 show that the AMN can solve a wide variety of machine reading tasks and that it behaves different from other memory networks.

7.4.2 Analysis

We analyze the hyperparameter settings used to produce the results in Table 8.4 and provide examples of the inner workings of the attention mechanism of the memory module.

Hyperparameters and speed of convergence

Table 7.2 lists the hyperparameter values for the smallest AMNs that achieve the best performance on the validation set, with fewest training examples. Here, *smallest network* refers to the size of the network in terms of embedding size and number of memories. The last column lists the number of batches needed. As can be seen from Table 7.2, AMNs can learn fast. As an example, it needs only 5 epochs to solve the first

Table 7.2: Hyperparameter values for the minimal AMNs that were fastest in achieving best performance on the validation set. The size refers to both size of embeddings and hidden states. The last column lists the number of batches needed.

Dataset	size	# layers	# mem	# batches
single supporting fact	32	1	1	1,000
two supporting facts	64	2	3	12,200
three supporting facts	64	2	3	14,000
two arg relations	32	1	1	1,200
three arg relations	32	1	2	3,000
yes-no questions	32	1	1	3,800
counting	32	1	3	5,000
lists sets	32	1	1	4,400
simple negation	32	1	2	3,200
indefinite knowledge	32	1	1	3,800
basic coreference	32	1	2	1,400
conjunction	32	1	1	1,200
comp coreference	32	1	1	10,000
time reasoning	64	2	1	6,000
basic deduction	32	1	1	2,200
basic induction	64	1	2	10,200
positional reasoning	32	1	3	6,200
size reasoning	32	1	3	2,400
path finding	64	1	1	13,000
agents motivations	32	1	3	3,600

dataset: there are 10k examples — 1,000 batches of 50 examples = 50k examples = 5 epochs. This is in contrast to the 100 epochs reported in [144] and 256 epochs listed as a maximum in [99]. Interestingly, adding depth to a network by stacking GRU cells was helpful in only 3 out of 20 cases.

Result analysis

Figures 7.2–7.7 show visualizations of the attention vectors of the memory module. The attention is visualized per memory step. Although some stories in the dataset are over 100 sentences in length, short examples were picked here, for reasons of brevity. Every column represents a memory step, and the values per memory step add up to 1 (barring rounding errors).

Figure 7.2 shows an example where one memory step is used. The attention focuses on the last time Daniel, the person the question is about, is mentioned. Interestingly, the second sentence also gets some attention, presumably because the bedroom, which features in the question, is being referred to. A particularly striking detail is that — correctly — nearly no attention is paid to the fifth sentence, although it is nearly identical to the question.

In Figure 7.3, attention is highest for sentences in which the person being asked

.000	mary got the milk there
.073	john moved to the bedroom
.000	mary discarded the milk
.000	john went to the garden
.004	daniel moved to the bedroom
.193	daniel went to the garden
.727	daniel travelled to the bathroom
.002	sandra travelled to the bedroom
.000	mary took the football there
.000	sandra grabbed the milk there

Figure 7.2: Dataset: yes-no questions, question: 'is daniel in the bedroom?', prediction: 'no', ground truth: 'no'.
The attention is visualized per memory step. Every column represents a memory step, and adds up to 1 (allowing for rounding errors).

.000	mary and daniel travelled to the bedroom
.000	then they journeyed to the hallway
.002	daniel and sandra went back to the garden
.384	following that they journeyed to the bathroom
.002	sandra and john went back to the bedroom
.000	then they journeyed to the garden
.000	john and daniel moved to the office
.025	after that they went back to the hallway
.001	sandra and daniel travelled to the bedroom
.587	after that they travelled to the hallway

Figure 7.3: Dataset: compound coreference, question: 'where is daniel?', prediction: 'hallway', ground truth: 'hallway'.

about is referred to. This is especially noteworthy, as the reference is only by a personal pronoun, which moreover refers to two people.

For the *size reasoning* dataset, three memory steps were needed (see Table 7.2). An example is shown in Figure 7.4. The first memory step mistakenly focuses on the sixth sentence about the chest. Gradually, however, the memory module recovers from this error, and attention shifts to the fourth sentence about the suitcase.

Figure 7.5 shows the ability of the network to focus only on relevant parts. Although the seventh and tenth sentence are nearly identical, it is the last sentence that matters, and it is this sentence the network attends to almost solely. Curiously, the two memory steps attend to the same sentences, which is consistently the case for this dataset. This might indicate that a single memory step could suffice too. Indeed, experiments show that on some datasets networks with fewer memory steps achieve the same or nearly the same performance as bigger networks, but take longer to reach it. The extra memory steps might serve as extra training material.

The last two cases, Figure 7.6 and 7.7, are from the *three supporting facts* dataset that the model could not solve. What stands out immediately is the fact that the attention is much more spread out than in other cases. This is the case throughout the entire dataset. It shows that the model is confused and fails to learn what is relevant. In

.001	.004	.005	the box is bigger than the chocolate
.036	.090	.105	the chocolate fits inside the suitcase
.024	.066	.080	the box is bigger than the box of chocolates
.216	.272	.296	the box of chocolates fits inside the suitcase
.052	.076	.080	the box is bigger than the box of chocolates
.458	.316	.275	the chocolate fits inside the chest
.120	.098	.090	the chocolate fits inside the box
.091	.075	.067	the box of chocolates fits inside the box
.001	.000	.000	the suitcase is bigger than the chest
.001	.002	.002	the suitcase is bigger than the chocolate

Figure 7.4: Dataset: size reasoning, question: 'is the suitcase bigger than the chocolate?', prediction: 'yes', ground truth: 'yes'.

.000	.000	bill moved to the bedroom
.000	.000	fred went to the hallway
.000	.000	jeff went to the garden
.000	.000	fred travelled to the office
.000	.000	mary took the apple there
.000	.000	mary passed the apple to bill
.000	.000	bill gave the apple to mary
.053	.045	mary passed the apple to bill
.000	.000	fred travelled to the bathroom
.940	.950	bill passed the apple to mary
.002	.002	bill went back to the office
.004	.003	mary dropped the apple

Figure 7.5: Dataset: three arg relations, question: 'what did bill give to mary?', prediction: 'apple', ground truth: 'apple'.

Figure 7.6 just reading the last five sentences would have been enough. The model does seem to capture that John picked up the apple, but only very weakly so. The crucial sentence, third from the end, is the sentence the model pays least attention to. Figure 7.6 shows the model being even more confused. It starts out by attending mostly to Mary, who has nothing to do with the story. The sentences that do matter, again, get very little attention.

Overall, these examples indicate that, when the AMN learns to solve a task, its memory module is very decisive in paying attention to the relevant parts of the input and ignoring the rest.

7.5 Conclusions

In this chapter the task of machine reading we addressed, a core task in natural language understanding and AI research. We introduced the Attentive Memory Network (AMN), an end-to-end trainable attention-based memory network with hierarchical input encoder, to answer the following research question:

..	..	..	...
.042	.043	.041	mary grabbed the apple
.032	.031	.030	john travelled to the hallway
.031	.029	.029	mary went back to the hallway
.040	.039	.038	sandra went back to the bedroom
.038	.036	.035	mary left the apple
.038	.035	.034	john dropped the milk
.049	.052	.051	john got the apple
.041	.041	.041	john dropped the apple
..	..	..	...
.045	.039	.040	john picked up the apple
.018	.014	.015	sandra went back to the garden
.006	.006	.007	john went back to the bedroom
.002	.002	.003	john went back to the bathroom
.002	.002	.002	mary moved to the garden

Figure 7.6: Dataset: three supporting facts, question: 'where was the apple before the bathroom?', prediction: 'garden', ground truth: 'bedroom'.
Columns do not add up to one as some (irrelevant) sentences were left out.

RQ4 Can an efficient memory network be designed using RNNs with attention mechanism only, without loosing performance?

AMNs solve 18 of 20 datasets designed specifically for evaluating algorithms on machine reading tasks. Analysis shows that they typically need only a few epochs to achieve optimal performance.

Our findings indicate that a simple architecture like the AMN is sufficient for solving a wide variety of machine reading tasks. Attention is a powerful mechanism that proves to have benefits outside of the machine translation setting it was originally introduced in.

The AMN architecture proposed in this chapter, involves a hierarchical input encoder that reads words and combines them into sentence representations, and was evaluated on English data. In the next chapter, we investigate if, when dealing with languages morphologically more involved than English, reading input at the level of bytes is to be preferred over reading words. Where we evaluated the AMN on 20 small synthetic datasets, made specifically for evaluating machine reading systems handling English data, in the next chapter we evaluate our models on three large scale datasets, based on real world texts written in three different, morphologically varied, languages.

..	..	..	...
.074	.045	.048	daniel travelled to the hallway
.121	.067	.075	mary travelled to the hallway
.070	.047	.049	mary went to the office
.050	.033	.033	sandra journeyed to the bathroom
.057	.037	.037	daniel took the milk
.054	.033	.036	daniel travelled to the kitchen
.018	.013	.015	mary moved to the bedroom
.025	.019	.021	daniel picked up the football there
.010	.011	.011	daniel journeyed to the office
.009	.010	.010	daniel left the milk there
.013	.015	.015	mary took the apple there
.006	.005	.006	sandra journeyed to the garden
.008	.008	.008	mary dropped the apple
.008	.009	.008	mary travelled to the kitchen

Figure 7.7: Dataset: three supporting facts, question: ‘where was the milk before the office?’, prediction: ‘hallway’, ground truth: ‘kitchen’.
Columns do not add up to one as some (irrelevant) sentences were left out.

8

Byte-level Machine Reading across Morphologically Varied Languages

8.1 Introduction

As discussed already in Chapter 1, natural language understanding is one of the long-standing goals of artificial intelligence, encompassing tasks like textual entailment [130], information extraction [39], semantic textual similarity [84], question answering [42] and machine reading [64]. In this chapter we focus on the latter task of machine reading, where a machine reads a document and answers questions about it. On English machine reading tasks, word-level models yield state-of-the-art results [31, 64, 65]. An advantage of English in this context is its relatively limited morphology that allows word-based models to get a broad coverage of word types actively used, while maintaining a manageable vocabulary size. However, in morphologically richer languages, e.g., Turkish, Russian, Finnish and Czech, many more word types exist, due to highly productive prefix and suffix mechanisms, and even very large vocabularies cannot provide extensive coverage. To illustrate, Figure 8.1 shows a Russian example in which two forms of the word *neuroradiologist* do not appear in a vocabulary of over 950,000 most frequent words. In a setting like English, with less morphological transformations, a word-level model with a pointer mechanism (see, e.g., [153]), that can copy strings verbatim from the input to the output, would be able to reproduce the unknown word, but in this case doing so would lead to the incorrect answer. Answering this query is therefore impossible for a word-level model. For a byte-level model, on the other hand, it is possible to reproduce the relevant word from the input and to apply the right morphological transformation, learned from similar training examples, even if the word itself was never observed during training. As another example, in Turkish, *kolay* means *easy*, *kolaylaştırabiliriz* means *we can make it easier*, while *kolaylaştıramıyoruz* means *we cannot make it easier*. Similarly, *zor* means *hard* or *difficult*, *zorlaştırabiliriz* means *we can make it harder*, while *zorlaştıramıyoruz* means *we cannot make it harder*. This example illustrates that a lot of semantic information is shared between words in Turkish, where in English separate words would be used. Hence, as in the Russian case, a larger vocabulary would be needed for Turkish compared to English to cover the same amount of text, which would increase the number of parameters of an embedding-based model dramatically. Learning these embeddings would, moreover, be hampered by

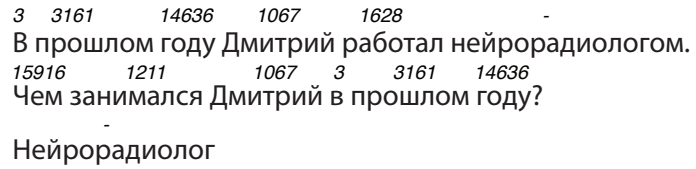


Figure 8.1: Example from the Russian machine reasing dataset. Document: *Dmitry worked as neuroradiologist last year.* Question: *What was Dmitry's occupation last year?* Answer: *Neuroradiologist.* Numbers are positions in a frequency ordered vocabulary of over 950K words.

fewer training examples being available per word type.

To overcome the issues caused by a lack of vocabulary coverage and by sparse training data outlined above, models have been proposed that work at the sub-word level, e.g., morphemes [25, 110]. Although morphemes are a natural unit for semantic composition, a drawback of these approaches is their dependency on high-quality morpheme segmentation algorithms, and the potential ambiguity of the morpheme segmentation itself. To avoid these complications, methods have been proposed that take characters as input [90, 107, 170]. An alternative approach is to consider bytes as input. Representing input as bytes is attractive as bytes provide a universal encoding format across languages. As such, reading bytes as input ties in with a long-standing ambition to learn language from scratch [170]. Moreover, representing input at the byte level allows for a small and fixed-size vocabulary of 256 tokens, which gives models a small memory footprint, compared to word-level models with tens or hundreds of thousands of words. A final advantage of using bytes over characters is that no character vocabulary has to be decided on (which can be non-trivial when dealing with real-world text like Wikipedia, where, e.g., Chinese characters, or characters of any other alphabet, can appear in any given article). Our goal in this research is to compare machine reading architectures across morphologically varied languages. Having an identical byte vocabulary across languages makes for better model comparisons, as there are fewer hyperparameters to choose (i.e., vocabularies and their sizes). To sum up, the research question we aim to answer in this chapter is:

RQ5 Is it advantageous, when processing morphologically rich languages, to use bytes rather than words as input and output in a machine reading task?

Using bytes as inputs to the models we consider allows us to compare models in an unbiased fashion, without introducing noise from a morphological decomposer.

Many different machine reading architectures have been proposed in literature [64, 76, 166]. Next to standard RNN sequence-to-sequence (seq2seq) models, convolutional RNNs [163, 170], word-character hybrid models [109] and memory networks [80, 120, 144, 158] have been proposed.[1] We implement 4 byte-level models, based on these families of models, and present a new seq2seq variant, called *encoder-transformer-decoder*. We evaluate all models on three large datasets: an al-

[1]Fully convolutional models for encoding textual data have been proposed [170] in text classification settings. Their performance in domains other than classification is not evident and we do not consider them in our experiments.

ready existing English dataset, and two additional datasets in Turkish and Russian, created specifically for this purpose. The two additional datasets are publicly available at `http://www.tomkenter.nl/index.php?page=byteLevelDataset`. All datasets are large enough to train neural models on. We compare results to the strongest word-level model available, and show that reading byte-level input is beneficial for all three languages considered.

Our main contributions are:

- We provide a platform for comparing machine reading models across different types of languages, by releasing 2 large machine reading datasets, one in Turkish, one in Russian — next to the already existing one in English.
- We implement 4 byte-level models based on the major families of machine reading models (vanilla RNN, convolutional RNN, hybrid word-byte-level, memory networks) and propose a seq2seq network variant, called encoder-transformer-decoder. It is the first time, to our knowledge, that multiple byte-level models are systematically compared on a single machine reading task, across fundamentally different languages.
- We show that for all three languages considered in the experiments, there are models reading bytes that outperform the current state-of-the-art word-level model.

We describe the datasets we introduce in §8.2. The models we use are described in §8.3, the experiments in §8.4, and results and analysis in §8.5 and §8.6, respectively.

8.2 Datasets and problem motivation

In [65] an English reading comprehension dataset is presented, called WikiReading. The set is constructed from Wikipedia and Wikidata [152] by constructing `(document, property, value)` triples, where the `property` and `value` originate from Wikidata triples, and the `document` is the original Wikipedia article text, as linked to in Wikidata. This process yields a challenging dataset as, unlike in other datasets, e.g., WikiQA [165], the values in the triples are not necessarily present in the Wikipedia document verbatim, and may have to be inferred from the text. In the experiments the `property` and `document` are provided as input to a reading comprehension algorithm, where the `property` is interpreted as being a query about the `document`, to which the `value` is the correct answer.

For the experiments on Russian and Turkish, we construct new datasets following the procedure described above. The size of every dataset is proportional to the size of the Wikipedia per language. The already existing English dataset is split in training/validation/test according to a 85/10/5 distribution. For the new sets, which are smaller, we choose a 80/10/10 split, to keep enough examples in the test set. Table 8.1 presents an overview of the number of examples (i.e., triples) for each dataset, where we list the numbers for the English dataset too for comparison. The sets are publicly available.[2]

[2]The Turkish and Russian datasets can be downloaded from `http://tomkenter.nl/index.php?page=byteLevelDataset`

Table 8.1: Number of examples per dataset. The bottom part lists the percentage of answers appearing verbatim in the document and the percentage of out-of-vocabulary tokens in the documents.

	English	Turkish	Russian
training	16.0M	655K	4.26M
validation	1.89M	81.6K	531K
test	941K	82.6K	533K
% verbatim	67.8	52.3	55.9
% OOV tokens	3.70	7.51	9.78

The bottom part of Table 8.1 indicates how the datasets differ in character. For the morphologically rich languages only approximately half of the answers appear in the documents. Furthermore, the documents in these languages contain up to three times as many out-of-vocabulary words as the documents in the English dataset. These numbers motivate exploring models that do not rely on word-level input only.

8.3 Models

Next to our baseline, we consider 5 encoder-decoder models in our experiments. The encoder-decoder architecture was proposed in a machine translation setting [32, 77, 145]. A sequence of symbols is encoded into a distributed representation, which is used as initial state of a recurrent decoder. In a translation setting, the input is an utterance in a source language, while the output is the same utterance, in a target language. In our present setting of reading comprehension, however, the input is twofold, a question and a document, while the output is an answer.

The key focus of our experiments is to distinguish between different ways of encoding the question and documents. Therefore, we apply a generic setup, where the encoder varies, while the decoder is the same for every model. Some building blocks are shared between models. For a discussion of RNNs and sequence-to-sequence models, we refer to Chapter 2. The encoder models are detailed in §8.3.1 and the decoder model in §8.3.2.

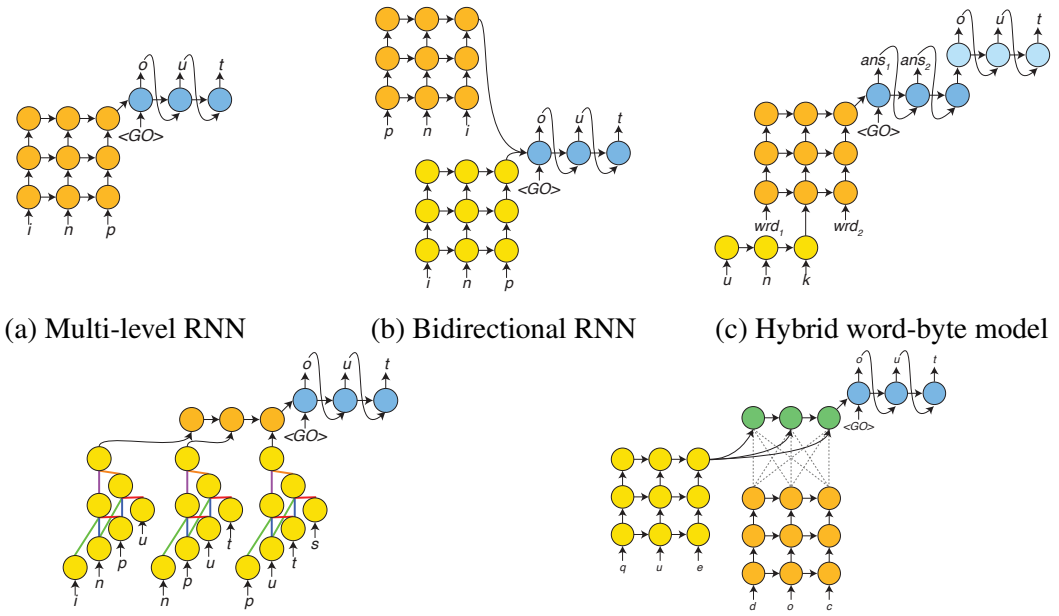
8.3.1 Encoders

Below we describe the encoder modules used in the experiments in §8.4.

Multi-level and bidirectional RNNs

The default way of encoding input in a sequence-to-sequence setup is to use an RNN. We test two commonly used variants, multi-level RNNs and bi-directional RNNs.

The multi-level RNN [32, 145], referred to as Deep Reader in [64], is an extension of a single-layer RNN encoder. Instead of having one recurrent cell, multiple cells are stacked and a separate set of parameters is maintained at every level. At level i, Equation 2.3 becomes $\theta_i^t = f(\bar{\mathbf{x}}_i^t, \theta_i^{t-1})$, where the input $\bar{\mathbf{x}}_0^t = \mathbf{x}^t$ and $\bar{\mathbf{x}}_i^t = \mathbf{h}_{i-1}^t$ for



(a) Multi-level RNN (b) Bidirectional RNN (c) Hybrid word-byte model

(d) Convolutional-recurrent (e) Memory network/encoder-transformer-decoder

Figure 8.2: Graphical illustration of the models employed (best viewed in color). The attention mechanism of the decoder, which is employed for all models, is left out for clarity.

$i > 0$. I.e., at every level but the first one, the input is replaced by the hidden state at the previous level. The multi-layer RNN encoder is illustrated in Figure 8.2a.

Bi-directional RNN Bidirectional RNNs (Figure 8.2b) have proven to be a robust choice in many different settings [64, 107]. The bidirectional RNN employs two multi-level encoders as described above: one reading the input from left to right, and one reading it from right to left. They yield two final states, $\overrightarrow{\mathbf{h}^t}$ and $\overleftarrow{\mathbf{h}^t}$, respectively. Following, e.g., [64], we concatenate the two states to get the final state of the two encoders combined: $\mathbf{h^t} = \overrightarrow{\mathbf{h}^t} \,||\, \overleftarrow{\mathbf{h}^t}$. If the size of the hidden states of the encoders is such that this yields a vector which size does not match the hidden state size of the next recurrent cell (in particular, it can be twice too big) we reshape: $\mathbf{h^t} = \mathbf{W}_{projection} \cdot \overrightarrow{\mathbf{h}^t} \,||\, \overleftarrow{\mathbf{h}^t}$.

Hybrid word-byte

As argued in §8.1, in morphologically rich languages many words might not appear in the vocabulary. An alternative to treating the input as either a stream of words or bytes is a hybrid approach, where the input is read word by word, and the model can resort to byte-level reading when a word is out-of-vocabulary. The word-byte hybrid model we employ follows the model presented in [109], specifically, the *separate-path* variant. Every time a word is encountered that does not appear in the word vocabulary, a byte-level encoder is run, whose last hidden state is used as distributed representation

of the word. As the recurrent cells following this initial layer are unaware of the source of a word representation (i.e., a word embedding matrix, or a byte-level encoder), the byte-level encoder learns to map words to the same embedding space the word vocabulary uses. At decoding time a similar procedure is employed, where a word-level decoder produces output, and a separate byte-level decoder is resorted to when a special 'unknown word' token is encountered. A graphical illustration of this model is provided in Figure 8.2c.

Convolutional-recurrent

The convolutional-recurrent model is based on the model in [163]. Convolutional filters are applied to the input byte embeddings, or directly to one-hot encodings. After multiple levels of convolutions a max-pooling layer is applied. This gives a fixed-size vector with as many elements as there are convolutional filters. Multiple of these fixed-size vectors are obtained from the input, the exact number depending on the input size, the receptive field of the convolutions (the window of input symbols they observe) and the stride. Each of these vectors is a distributed representation of a window of input symbols. A recurrent encoder is used to encode this sequence of representations into its final state. Figure 8.2d provides a graphical representation of this model. An alternative approach, already discussed in §3.3.2, is to take word boundaries into account [76]. Preliminary experiments showed inferior performance and we leave this variant out of our experiments.

Memory networks

There are two main differences between memory networks [80, 120, 144, 158] and the standard encoder-decoder architecture: 1) a number of recurrent steps is performed between encoding and decoding 2) there are two separate input encoders – one for the question and one for the document.

The final state of the document encoder, before being presented to the decoder, is modified during a number of recurrent steps (also referred to as memory hops), conditioned on the final state of the question encoder. Specifically, at every time step, a recurrent cell attends over the hidden states of the document encoder, and is provided with the final state of the question encoder as input (i.e., it has the same input at every time step). That is $\mathbf{x}^t$ in Equation 2.3 is $\mathbf{h}^n_{question}$, the last hidden state of the question encoder, for a query of length n. The states to attend over for the memory module, $\mathbf{H}_a$ in Equation 2.5, are $\mathbf{H}_{document}$, the hidden states of the document encoder. Finally, a crucial part of the memory network architecture is that the attention of the decoder is on the memory states. Figure 8.2e shows a graphical illustration of the memory network. The attention of the memory cells is represented by grey dashed lines.

Encoder-transformer-decoder

We present a variant of the memory network architecture, which differs in that the decoder attends over the document encoder states. Although this is a small deviation in terms of architecture, the model in fact now learns something crucially different. Where, in the memory network scenario, the memory cells can copy parts of the input

Table 8.2: Hyperparameter values tuned over

Embedding size	128/256
Internal state size	256/512
Number of RNN cells stacked	1/2
RNN cell type	LSTM/GRU
Input/output embedding tying	yes/no
Gradient clipping	0.1/1
Learning rate	$10^{-5}/10^{-4}/10^{-3}$

document based on the question, in the new setup, the internal states of the intermediate RNN do not serve as memories, because the decoder is in fact agnostic to them. Rather, the function of the intermediate RNN is to optimize the final state of the document encoder for the decoder, with respect to the question. As this model is similar to, but different from, the recently proposed encoder-reviewer-decoder model [166], we refer to it as *encoder-transformer-decoder* model. Since the key difference between this model and the memory model is in the attention of the decoder, which is not displayed in Figure 8.2, the graphical depiction of these models is shared in Figure 8.2e.

See §3.3.2 for a discussion of the models presented here and the existing models.

8.3.2 Decoder

Except for the word-byte hybrid model, we use the same decoder for all models, which is a single level RNN. At every time step a softmax over the byte vocabulary is computed, and the cross entropy is used to compute the loss. The decoder always applies attention to the hidden states of the document encoder, except for the memory model, where it attends over the hidden states of the memory cells (§8.3.1). In all experiments, when an LSTM cell is used in the encoder, an LSTM cell is used in the decoder, and likewise for GRUs.

8.4 Experimental setup

Table 8.2 lists the values of hyperparameters tuned over on the validation data. For the multi-level, bidirectional and convolutional-recurrent encoder, the document is appended to the query with a separator symbol in between. All embeddings are trained from scratch (i.e., no pre-trained vectors are used). We experiment with either sharing input and output embeddings, i.e., a single embedding matrix is employed, or having two separate embedding matrices [125]. For the memory network and the encoder-transformer-decoder model, the intermediate RNN performs 2 recurrent steps, as this yielded consistent performance in preliminary experiments.

At most 50 bytes are read of each question, and at most 400 bytes from the documents. The word-byte hybrid model observes roughly the same amount of input — 60 words per document, which, given the average word length in English, is ~ 400 characters. The other languages have longer words, so this model might have the

Table 8.3: Filter and maxpool width per layer for the convolutional RNN. For additional details, see [163].

	# layers	Filter width	maxpool width
C2R1	2	5, 3	2, 2
C3R1	3	5, 5, 3	2, 2, 2
C4R1	4	5, 5, 3, 3	2, 2, 2, 2
C5R1	5	5, 5, 3, 3, 3	2, 2, 2, -, 2

advantage of reading slightly more input on average. The maximum number of output steps is 50, which is enough for all answers in the training and validation set. Table 8.3 lists the additional hyperparameters tuned over for the convolutional-recurrent model.

All models are trained with stochastic gradient descent. The learning rate is adapted per parameter with Adam [91]. Batch size is 64. After every 50,000 batches, the learning rate is divided by 2.

Baseline model We compare the performance of our byte-level models to the model performing strongest on the English dataset in [65], which is a word-level sequence-to-sequence model with LSTM cells with state size 1024, a word vocabulary of 100,000, and 300d embeddings. The model employs placeholders to handle out-of-vocabulary words. To keep comparison between models as clean as possible, we do not pre-train the embeddings as in [65].

Evaluation An answer is considered to be correct if it is exactly the same as the ground truth answer (there is no stemming and no normalization, except for dates, which were all converted to one format, like *1 January 1970*). As some questions have a set of values as an answer (e.g., *Children of person X*), we compute precision and recall for every example and use the *mean F1* over all examples as evaluation metric, following [65].

8.5 Results

Table 8.4 lists the main results of the experiments. The first observation from the results is that for every language, there is a model reading bytes that outperforms the state-of-the-art word-level model. Interestingly, there are differences between models across datasets which can be explained by the characteristics of the languages in the dataset.

On the dataset of the morphologically most involved language we consider, Turkish, the difference between byte-level and word-level models is most pronounced. Here, all byte-level models outperform the word-level model, except for the convolutional RNN. This results seems to corroborate the hypothesis stated in §8.1, that a word-level model has trouble getting enough coverage in this language. The word-byte hybrid model lags behind here, compared to the other datasets. This is likely to be caused by the larger number of unknown words in the Turkish dataset (cf. Table 8.1). In English, copying words from the input document works, as demonstrated already by the word-level placeholder model being the best performer reported in [65]. This indicates that the unknown words might be, e.g., names and proper nouns, occurring in the text.

Table 8.4: Results in terms of mean F_1 of all models on all three datasets.

	Turkish	Russian	English
Multi-level RNN	**0.6956**	0.5784	0.7176
Bidirectional RNN	0.6627	0.5431	0.6615
Convolutional RNN	0.5753	0.4123	0.5364
Hybrid word-byte	0.6654	**0.5874**	**0.7418**
Memory network	0.6899	0.5612	0.7176
Encoder-transformer-decoder	**0.6956**	0.5808	0.7182
Word-level	0.6365	0.5759	0.7365

In Turkish, however, due to the morphological richness, many more words, not just proper nouns, are out-of-vocabulary, but there is relatively little data per word to learn from.

Russian, being highly inflective rather than agglutinative like Turkish, is morphologically less rich. This is reflected in the results, where only two byte-level models outperform the word-level model, and only marginally. The encoder-transformer-decoder model, while not the top performer, does beat the word-level baseline.

We should note that, although the improvements of the byte-level models over the word-level models seem small, the model sizes of the byte-only models are considerably smaller, as they have a vocabulary of 256 rather than 100,000. As such, the strongest byte-only model across the datasets of morphologically rich languages, the encoder-transformer-decoder model, yields two crucial benefits over the state-of-the-art word-level model: improved performance while having a substantially smaller memory footprint. This leads us to answer RQ5 *Is it advantageous, when processing morphologically rich languages, to use bytes rather than words as input and output in a machine reading task?* affirmatively.

Finally, on the English data, we do not expect the byte-level models to have an advantage over the word-level model, which is confirmed by the results in Table 8.4. The exception to this rule is the word-byte hybrid model. Most notably, not only does it outperform the word-level model in Table 8.4 — which has access to 60 words — it also outperforms the best performing model as reported in [65] which reads 300 words, and has a score of 0.718. This shows that having access to more information might actually hamper a model. More importantly, it indicates that reverting to reading bytes is a better way of dealing with out-of-vocabulary words than using placeholders.

There are multiple unexpected results in Table 8.4. Most remarkably, bidirectional models do not perform better than uni-directional RNNs. This is a surprising deviation from previous research, which has shown them to be consistent performers. The results indicate that on the byte-level, an extra backward pass does not yield additional, complementary information. Next, it is not clear from scrutiny of the data, why the scores on the Russian data should be lower in general (roughly 10–15% for each model). Additional analysis is necessary to disclose the underlying reason.

To sum up, the results on the three datasets show that byte-level input is beneficial for all three language considered. Moreover, the results indicate that the language characteristics play a pivotal role in determining the most appropriate model per lan-

Table 8.5: Results in terms of mean F_1 on categorical queries.

	Categorical		
	Turkish	Russian	English
Multi-level RNN	0.8528	0.7388	0.8482
Bidirectional RNN	0.8420	0.7289	0.8332
Convolutional RNN	0.8059	0.7082	0.7794
Hybrid word-byte	0.8478	**0.8022**	**0.8656**
Memory network	**0.8576**	0.7369	0.8595
Encoder-transformer-decoder	0.8539	0.7427	0.8461
Word-level	0.8125	0.7513	0.8531

Table 8.6: Results in terms of mean F_1 on relational queries.

	Relational		
	Turkish	Russian	English
Multi-level RNN	**0.5453**	0.4080	0.5752
Bidirectional RNN	0.4906	0.3524	0.4594
Convolutional RNN	0.3539	0.1206	0.2385
Hybrid word-byte	0.4903	0.3833	0.5824
Memory network	0.5290	0.3769	0.5561
Encoder-transformer-decoder	**0.5437**	**0.4089**	0.5749
Word-level	0.4676	0.4019	**0.5875**

guage. It is interesting to see that the advantage of reading bytes versus words appears to be proportional to the morphological complexity of the languages considered. On the English dataset, only one byte-level model outperforms the word-level baseline — the best scoring model reported in [65]. On the inflective language data, the Russian dataset, multiple byte-level models improve over the baseline, while on the data for the agglutinative language, Turkish, all byte-level models but one do.

8.6 Analysis

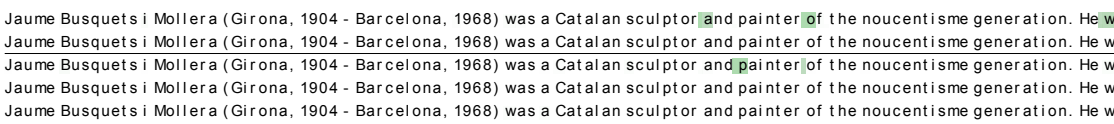
Results per query type Some properties, like *gender* or *instance of* are category-like, while some, like *mayor*, are more open-ended. We split out the results in Table 8.4 per query type, where we distinguish between *categorical* (few possible answers) and *relational* (a large set of possible answers). As dates are notoriously hard to deal with, we also split out a separate *date* category. The results for categorical queries, relational queries and date queries are displayed in Tables 8.5, 8.6, and 8.7, respectively.

A key observation is that the encoder-transformer-decoder model is a consistent (near) top performer across categories. Interestingly, on the Turkish dataset, the most pronounced difference between the well performing byte-level models and the word-level model is in the hardest category, the relational queries.

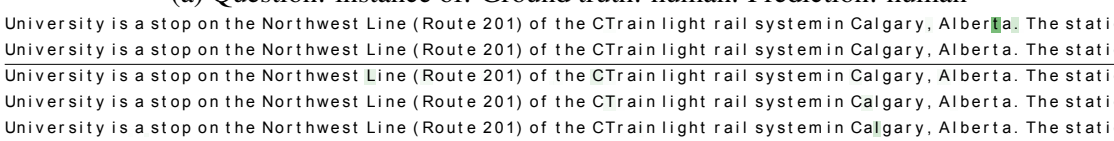
Visualization of encoder-transformer-decoder network As noted above, the encoder-

Table 8.7: Results in terms of mean F_1 on date queries.

	Dates		
	Turkish	Russian	English
Multi-level RNN	0.7796	0.7253	0.8075
Bidirectional RNN	0.7658	0.7142	0.7984
Convolutional RNN	0.4857	0.1039	0.3891
Hybrid word-byte	0.6784	**0.7507**	0.8000
Memory network	0.7740	0.7147	0.8067
Encoder-transformer-decoder	**0.7822**	0.7276	**0.8093**
Word-level	0.6218	0.7305	0.8021



(a) Question: instance of. Ground truth: human. Prediction: human



(b) Question: country. Ground truth: Canada. Prediction: Canada

Figure 8.3: Visualization of attention vectors of encoder-transformer-decoder model. Attention over the relevant part of the input is displayed for the 2 intermediate RNN steps (first 2 lines) and the decoder steps (next 50 lines, of which only the first 3 are shown for brevity).

transformer-decoder model is a top or near top performer on both morphologically rich languages. To gain insight in its inner workings, we visualize its attention vectors for two examples, in Figure 8.3. For ease of understanding, the examples are from the English dataset. Note that, since the inputs are bytes, the attention is typically at the end of a word (when all the bytes of a word have been read).

In Figure 8.3a, the answer does not appear in the document but has to be inferred. The first transformer step focusses on words indicating a human actor, like *sculptor*, *painter* and *he*. Figure 8.3b shows an example of the first transformer step solving the answer completely. The attention is on *Calgary, Alberta* which is enough, apparently, to infer the corresponding country. In both cases, the intermediate RNN steps, by focussing on the crucial parts of the input and altering their internal state accordingly, appear to provide the decoder with all the information it needs to generate the answer, which results in the decoder attention being very weak.

Hyperparameters As to the hyperparameters tuned over (Table 8.2) some patterns could be discerned, the most notable one being that GRUs proved to be better than LSTMs in the majority of cases. Furthermore, the learning rate is a crucial factor, which is less surprising. Tying the input and output embeddings typically helped. Embedding and state sizes did not matter greatly, nor did the various values for gradient clipping.

8.7 Conclusions

In this chapter we addressed the following research question:

RQ5 Is it advantageous, when processing morphologically rich languages, to use bytes rather than words as input and output in a machine reading task?

We introduced two large-scale reading comprehension datasets of morphologically rich languages, Turkish and Russian, which are publicly available. The new datasets, together with the English dataset published previously, provide a unique collection for comparing machine reading algorithms on one task, across principally different languages. We provide results for byte-level implementations of four major architectures of machine reading models. Furthermore, we introduce an encoder-transformer-decoder network model that performs best on Turkish data, and is competitive on the Russian data. It is the first time, to our knowledge, that multiple byte-level models are systematically compared on a single machine reading task, across fundamentally different languages. We show that on all datasets, reading input at byte level is beneficial, and that the encoder-transformer-decoder model is top or near top performer on the datasets of the morphologically more involved languages. The differences in performance between models across the datasets indicate that machine reading on different kinds of languages requires different architectures. It is interesting to see that the advantage of reading bytes versus words seems to be proportional to the morphological complexity of the languages considered. In the English dataset, only one byte-level model outperforms the word-level baseline. On the inflective language data, the Russian dataset, multiple models do, while on the data for the agglutinative language, Turkish, all byte-level models, but one, outperform the best scoring model originally reported on the English dataset. More research is needed to confirm whether this trend holds more broadly.

This chapter was the last of the research chapters in this thesis. Next, we conclude our work in natural language understanding at the level of words, short texts and documents.

9 Conclusions

In this thesis, we studied multiple aspects of the general task of automatically understanding text. We did so at the level of words in Part I, short texts in Part II and full documents in Part III.

In the next section, we discuss the main findings of the research presented in Chapters 4, 5, 6, 7 and 8. Subsequently, in §9.2, we will discuss limitations and directions for future research.

9.1 Main findings

Words

In the first part we studied changes in word usage over time. People in different times use different words to refer to similar underlying concepts. History scholars who have extensively studied a particular period in time may be knowledgeable about the vocabulary used in that period to refer to certain concepts. It is inconceivable, however, for a single person to be aware of the shifts in vocabulary that happened across all periods in time, regarding arbitrary subjects. Therefore, in Chapter 4, we proposed a automatic way of monitoring vocabulary shifts over time, in an ad hoc search session, i.e., given arbitrary input. We presented multiple algorithms for monitoring vocabularies over time and performed systematic evaluation of their results. Our results show that our approach of combining an exploratory method of generating shifting vocabularies over time with a conservative approach yields consistent performance.

Short texts

In the second part of this thesis, text understanding at the level of short texts was studied. Research in this area can be divided into two subfields, where in one field, methods are employed that use pre-defined features, optimally combined for the task at hand, while in the second field, end-to-end trainable models are developed, which try to learn a specific task from training data only. We presented work pertaining to both strands of research.

In Chapter 5 multiple sets of pre-trained word embeddings were leveraged to capture semantic similarities between two short texts, represented as features to be used

by a learning algorithm predicting semantic similarity. Our generic, semantics-only method of computing semantic similarity between short texts outperforms all baseline approaches working under the same conditions, and more interestingly also exceeds all approaches using external sources of structured semantic knowledge that, to our knowledge, were evaluated on this dataset. This implies that distributional semantics has come to a level where it can be employed by itself in a generic approach for producing features that can be used to yield state-of-the-art performance on the short text similarity task, even if no manually tuned features are added that optimize for a specific test set or domain. This is particularly promising, considering no supervised material is needed, since for word embeddings to be trained, only a large body of text is required, which should be available even for low-resource languages.

An alternative approach to using pre-trained word embeddings was taken in Chapter 6. Here, word embeddings were trained from scratch, and were optimized specifically for the task of being aggregated to represent the semantics of short texts. Our results show that, although averaging generic word embeddings, not trained for a particular task, to represent short tasks can yield strong baseline, optimizing the embeddings for this task can be beneficial in a large variety of settings. A similar phenomenon may occur across other tasks, such as optimizing embeddings to represent entities or user queries in an IR system.

Documents

The third and final part of this thesis is about text understanding of documents. While the semantics of words can be meaningfully expressed in terms of relations to other words (an intuition used both in WordNet and in distributed representations such as word2vec word embeddings), it is not clear this should work for full documents too. Although automatically determining semantic similarity between documents is a challenging task in itself [100, 104, 163] the interplay between words, statements and implicitly conveyed information in documents gives rise to more intricate semantic phenomena than can be captured by document similarities only. One way of determining whether a computer program understands a text at the level of full documents is to determine whether it can answer questions about it. This task, which is commonly referred to as machine reading or reading comprehension, is studied in the Chapters 7 and 8.

In Chapter 7 the Attentive Memory Network architecture was presented. Different from previous memory networks, which construct internal memory states directly from word-level input, the Attentive Memory Network uses a hierarchical input encoder, that first computes sentence representations from input words, and constructs internal memory states only from the sentence representations. This hierarchical procedure allows for a more efficient processing of input documents, which is the main advantage of the Attentive Memory Network over alternative approaches.

Finally, in Chapter 8, text understanding of documents was conducted in morphologically varied languages. Where in previous chapters, which dealt with English texts only, words were a natural choice of granularity of input, in languages morphologically richer than English, this choice is less evident. For example, in inflective languages like Russian, and agglutinative languages like Turkish, substantially more unique words exist than in English. This hampers the training of word embeddings, as there are too many

word types to train embeddings for efficiently and, moreover, there are fewer examples per word type to learn from. Therefore, in this chapter, we proposed to read input in bytes. Bytes have the additional advantage of providing a universal encoding for all languages, making them a particular suitable choice of input unit when multiple models are to be compared across language, as we did. We show that on datasets in three languages of different morphological complexity, Turkish, Russian and English, reading bytes as input is beneficial. Furthermore, we show that that the encoder-transformer-decoder model we introduce is a top or near top performer on the datasets of the morphologically rich languages.

The Turkish and Russian machine reading datasets we publish, together with the English dataset published previously, provide the first data collection available, to our knowledge, for comparing machine reading algorithms on one task, across principally different languages. We hope that the new datasets encourage the development of novel approaches for machine reading in morphologically rich languages, especially since the results on them are still considerably lower compared to the results on English counterpart.

9.2 Future research directions

In this section we discuss limitations of the research discussed in this thesis, and propose directions of future research.

Words

As mentioned in §1.1 word-level semantics have been studied for many years. Where earlier work focussed on manually created resources of semantic information [121], methods based on corpus statistics have gained popularity over the last decades [37, 40, 118, 119, 124]. As methods based on corpus statistics can be used for quick and computationally cheap analysis of large bodies of text, they allow for ambitious questions to asked about these texts, including the ones about changes in word usage we studied in the first parts of this thesis.

Ad Hoc Monitoring of Vocabulary Shifts over Time

Since the task of studying changing vocabularies is, to the best of our knowledge, a new task, it brings new questions with it. From a digital humanities perspective, e.g., it would be interesting to see whether changes in word usage in particular periods in time could be tied to socio-economic events preceding the changes. From a computational perspective, the temporal aspect provides an interesting challenge. Time is not explicitly modeled in embedding models, which is the reason a series of embedding models is used in Chapter 4. A promising direction for future research would be to incorporate the time dimension itself in an embedding model.

The dimension of time also has implications concerning evaluation, as the performance of an adaptive method for monitoring shifting vocabularies may degrade or improve over time. However, traditional evaluation metrics like NDCG or MAP are time-agnostic. Additional insights could be obtained when a time-aware evaluation

metric, such as, e.g., proposed in [82] in the context of document filtering systems, would be applied to the present setting.

Future work on monitoring vocabulary shifts over time could focus on longer evaluation periods than we did in our experiments in Chapter 4, e.g., a century of material. However, additional work on making the methods more efficient, without losing performance, would be needed for this to work. In our work, as many word embedding spaces were computed as there are years in the period under consideration, as described in §4.2. Computing semantic similarities between words in all of these spaces, as is needed to construct the semantic graphs is computationally expensive. Additionally, storing the embedding spaces on disk and loading them all into memory puts demanding constraints on the hardware needed to run the method.

High-quality, on-topic vocabularies over time can be beneficial in many cases both in IR and in digital humanities research. The vocabularies can be used as a way of exploring data, as is the underlying scenario in our research in Chapter 4. Furthermore, as mentioned above, they could be used for time-aware query expansion, where the query expansion depends on the timestamps of documents in a corpus. An application like this could partially reverse the roles of researchers and search tool. Where currently history scholars have to be well-versed already in the period they study to be able to find documents they are looking for with a time-agnostic search engine, if a search engine with time-aware query expansion would be available, they might instead learn from the tool. I.e., currently, to find historic documents pertaining to a certain topic, one would have to already know the right keywords that describe the topic in the intended period in time. However, with high-quality time-aware query expansion, a scholar could learn these very terms themselves from the tool.

Lastly, different types of shift in vocabulary might be discerned. Similar to how document ranking systems are tailored towards query intent, systems for monitoring shifting vocabularies over time could be optimized in terms of optimal parameter settings or choice of algorithm, depending on the type of vocabulary shift they aim to monitor.

Short texts

Determining whether two short text fragments have a similar meaning is usually not a task on its own in real-world scenarios. Rather, the research in this area is typically incorporated in downstream applications.

Short Text Similarity with Word Embeddings

Algorithms for learning short text similarity could be used in automatically created probabilistic knowledge bases, as described in, e.g., [39]. In this work, triples are extracted from an input corpus and have a confidence score associated with them based on the number of sentences in the corpus describing the relation in the triple. If short text similarity can be reliably determined, multiple sentences that support the relation expressed in the triple, but which are phrased differently, can be recognized as corroborating evidence for it. In this way, short text similarity can be used to improve this confidence score.

An evident limitation of calculating meta-features in the manner we propose, i.e., from averaged word vectors and word alignments, is that the order of words is not taken into account. This goes for any bag-of-words model, of course. The reason the difference between word-order-aware models and bag-of-word models is not apparent from the results on sentence similarity tasks over the years might be that the commonly used evaluation sets do not contain enough sentence pairs (if at all) in which word order is a crucial factor.

By evaluating our method of determining semantic similarity between short texts on sentence pairs in §5.3, we measured its efficacy directly. However, there is no constraint, theoretical or other, that limits the method to being applied to short texts. It would be interesting to see how other fields of research that deal with large corpora of unstructured texts can benefit. An example would be to use our method of determining semantically similarity between texts to find candidate suspicious documents in a plagiarism detection system.

Siamese CBOW

Word and sentence embeddings are ubiquitous and many different ways of using them in supervised tasks have been proposed. It is beyond the scope of this section to provide a comprehensive analysis of all supervised methods using word or sentence embeddings and the effect Siamese CBOW would have on them. However, it would be interesting to see how Siamese CBOW embeddings would affect results in supervised tasks, like, e.g., sentiment analysis, where short text fragments play a crucial part in determining the overall sentiment.

Word types that do not occur during training time, or that occur infrequently, are problematic for the Siamese CBOW model, much as for any other co-occurrence-based embedding method like word2vec or GloVe. An efficient way of folding newly observed words into an already existing embedding space would be a valuable addition to our approach.

Although we evaluated Siamese CBOW on sentence pairs, there is no theoretical limitation restricting it to sentences. It would be interesting to see how embeddings for larger pieces of texts, such as documents, would perform in document clustering or document filtering tasks.

Lastly, if the model would allow for two distinct sets of words embeddings — a minor variation on the architecture presented in Chapter 6, where input embeddings and output embeddings would correspond to different vocabularies — and a parallel corpus would be available, Siamese CBOW could be used to learn word embeddings which are translations of one another (as in [52]) and could provide an efficient way to compare sentences in different languages.

Documents

As noted at the start of Chapter 1, digital assistants which recently became available set ambitious goals and expectations for natural language understanding algorithms. The research about the machine reading task addressed in the third part of this thesis aims to contribute to these ambitions.

Attentive Memory Networks for Natural Language Understanding

Memory networks have been applied in settings where external knowledge is available, in particular in the form of key-value pairs [120]. Although this setting is different from the machine reading setting, it would be interesting to see how Attentive Memory Networks could be applied as a key-value memory networks. This is a particularly promising direction considering the large amounts of data available as triples, such as DBpedia.[1] A memory network trained on this data could be used in a conversational system which knows about every Wikipedia fact available.

Instead of using the memory in the memory network to store facts, in dialogue systems, a different way of using memory is possible. The context of the conversation, including the utterances produced by the system itself, could be stored as memory, and taken into consideration before generating every next utterance. This could be one step towards solving the problem with consistency that many conversational agents have.

In current machine reading research, single documents are typically used as a reference. An interesting additional direction would be to investigate how multiple documents can be taken into account. For example, could all newspaper articles published in particular time frame be used? The current architectures do not allow for multiple inputs, and moreover, the size of all input combined would be too large to store internally in the model. Future research might shed light on what the components are in the current approaches that prohibit us from dealing with large input sets, and how they should be modified.

Byte-level Machine Reading across Morphologically Varied Languages

Having a small unit of inputs, such as bytes, means that the encoder RNN of any sequence-to-sequence model has to be unrolled for a larger number of steps compared to when, e.g., the input is at word level. As the input documents become larger, the large number of unrolling steps can become computationally prohibitive. This currently is a limiting factor in our approach, though directions for solutions are available, such as hierarchical input encoders.

Since bytes provide a universal encoding across languages, it should be interesting to investigate whether learning can happen across different languages simultaneously. That is, if training material is presented in different languages, could a single byte-level model learn to read and comprehend documents, and answer questions about them, regardless of the language they are written in?

Finally, previous research has treated information retrieval in historic language as a cross-language approach [96]. Similar to learning machine reading systems across languages, an attempt could be made at training a system on text material published throughout a century. This relates to the research described in the first research Chapter 4, where differences in word usage over time were studied. A machine reading system trained from byte level up on historic material might pick up the gradual shifts in spelling and word usage occurring throughout the corpus, and by doing so might learn to answer

[1] DBpedia is a structured version of Wikipedia, storing the information contained in Wikipedia as triples. See `http://wiki.dbpedia.org/`.

questions about historic documents regardless of the specific words and phrases used throughout time.

9.3 Final outlook

Text understanding today is at an unprecedented level. It plays a role in many tasks, among which are dialogue systems, or conversational agents. For the first time in history people can have more or less meaningful conversations with digital assistants, such as the ones mentioned in Chapter 1. While conversational systems used to be toy systems in a lab, nowadays virtually everyone carries one with them on their phone, and Amazon Alexa and Google Home devices are being sold in non-trivial numbers.

Still, the conversational systems we have today are miles away from omniscient systems like the Star Trek computer or any other hyper-intelligent AI envisioned in science fiction, and they will probably not pass the Turing test any time soon. However, they can perform mundane tasks like setting an alarm clock, making appointments in a personal calendar, switching on the lights, playing a television show, or reporting what the weather is going to be like, all simply by being told to do so in natural language. As new functionalities are being added, time will tell which ones will catch on, a process itself influencing how we will interact with computers. Do we stick to them being personal assistants that can do simple tasks for us? Or do we end up having full-fledged conversations with them, like the one the bots on the cover of this book seem to be engaged in?

A
Resources

Part of the contribution of this thesis is a collection of resources that were made available. This includes software code as well as data. Specifically, the resources are:

- The ground truth material used to obtain the results reported in Chapter 4 Ad Hoc Monitoring of Vocabulary Shifts over Time can be downloaded from `https://bitbucket.org/TomKenter/ad-hoc-monitoring-of-vocabulary-shifts-over-time-ground-truth`
- The code implementing the Siamese CBOW model as described in Chapter 6 Siamese CBOW can be downloaded from `https://bitbucket.org/TomKenter/siamese-cbow`
- The code implementing the Attentive Memory Network as described in Chapter 7 Attentive Memory Networks for Natural Language Understanding can be downloaded from `https://bitbucket.org/TomKenter/attentive-memory-networks-code`
- In Chapter 8 two machine reading datasets were introduced. Both datasets are released publicly and can be downloaded from `http://tomkenter.nl/index.php?page=byteLevelDataset`.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2015. (Cited on page 89.)

[2] E. Agirre, M. Diab, D. Cer, and A. Gonzalez-Agirre. SemEval-2012 task 6: A pilot on semantic textual similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 385–393, 2012. (Cited on pages 3, 23, 62, and 75.)

[3] E. Agirre, D. Cer, M. Diab, A. Gonzalez-Agirre, and W. Guo. Sem 2013 shared task: Semantic textual similarity, including a pilot on typed-similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task (*SEM 2013)*, pages 32–43, 2013. (Cited on pages 23, 53, and 62.)

[4] E. Agirre, C. Banea, C. Cardie, D. Cer, M. Diab, A. Gonzalez-Agirre, W. Guo, R. Mihalcea, G. Rigau, and J. Wiebe. SemEval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 81–91, 2014.

[5] E. Agirre, C. Banea, C. Cardie, D. Cer, M. Diab, A. Gonzalez-Agirre, W. Guo, I. N. Lopez-Gazpio, M. Maritxalar, R. Mihalcea, G. Rigau, L. Uria, and J. Wiebe. SemEval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 252–263, 2015. (Cited on pages 3 and 75.)

[6] R. M. Aliguliyev. A new sentence similarity measure and sentence based extractive technique for automatic text summarization. *Expert Systems with Applications*, 2009. (Cited on pages 3 and 53.)

[7] J. Allan. *Topic detection and tracking: event-based information organization*. Springer Science & Business Media, 2002. (Cited on page 20.)

[8] P. Annesi, D. Croce, and R. Basili. Semantic compositionality in tree kernels. In *Proceedings of ACM International Conference on Information and Knowledge Management (CIKM 2014)*, 2014. (Cited on pages 22, 54, 62, and 64.)

[9] H. Azarbonyad, M. Dehghani, T. Kenter, M. Marx, J. Kamps, and M. de Rijke. Hierarchical re-estimation of topic models for measuring topical diversity. In *Proceedings of the 39th European Conference on Information Retrieval (ECIR 2017)*, 2017. (Cited on page 8.)

[10] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR 2015)*, 2014. (Cited on pages 1 and 16.)

[11] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(5):54–63, 2016. (Cited on page 75.)

[12] C. Banea, D. Chen, R. Mihalcea, C. Cardie, and J. Wiebe. SimCompass: Using deep learning word embeddings to assess cross-level similarity. *SemEval 2014*, 2014. (Cited on pages 23, 55, 57, 63, and 66.)

[13] D. Bär, C. Biemann, I. Gurevych, and T. Zesch. UKP: Computing semantic textual similarity by combining multiple content similarity measures. In *SemEval 2012*, 2012. (Cited on pages 2, 3, 23, and 53.)

[14] M. Baroni, G. Dinu, and G. Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the The 52th Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, 2014. (Cited on pages 21 and 60.)

[15] P. Bellot, G. Kazai, M. Preminger, M. Trappett, A. Doucet, M. Koolen, E. SanJuan, A. Trotman, S. Geva, A. Mishra, R. Schenkel, M. Sanderson, S. Gurajada, V. Moriceau, X. Tannier, F. Scholer, J. Kamps, J. Mothe, M. Theobald, and Q. Wang. Report on INEX 2013. *SIGIR Forum*, 47(2):21–32, 2013. (Cited on page 60.)

[16] R. Berendsen, E. Meij, D. Odijk, M. de Rijke, and W. Weerkamp. The University of Amsterdam at TREC 2012. In *Proceedings of the Twenty-First Text REtrieval Conference (TREC 2012)*, 2012. (Cited on page 20.)

[17] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM review*, 41(2):335–362, 1999. (Cited on page 3.)

[18] A. Betti and H. van den Berg. Modelling the history of ideas. *British Journal for the History of Philosophy*, 2014. (Cited on pages 20 and 34.)

[19] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python*. O'Reilly Media, Inc., 2009. (Cited on pages 40 and 61.)

[20] D. M. Blei and J. D. Lafferty. Dynamic topic models. In *Proceedings of the 23rd international conference on Machine learning*, 2006. (Cited on page 19.)

[21] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003. (Cited on page 3.)

[22] D. M. Blei, T. L. Griffiths, and M. I. Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *J. ACM*, 2010. (Cited on page 19.)

[23] A. Bordes, Y.-L. Boureau, and J. Weston. Learning end-to-end goal-oriented dialog. In *Proceedings of the International Conference on Learning Representations (ICLR 2017)*, 2017. (Cited on page 85.)

[24] A. Borisov, I. Markov, M. de Rijke, and P. Serdyukov. A neural click model for web search. In *Proceedings of the 25th International Conference on World Wide Web*, pages 531–541, 2016. (Cited on page 3.)

[25] J. A. Botha and P. Blunsom. Compositional morphology for word representations and language modelling. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, 2014. (Cited on page 98.)

[26] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International World-Wide Web Conference (WWW 1998)*, 1998. (Cited on pages 36 and 49.)

[27] A. Budanitsky and G. Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 2006. (Cited on page 4.)

[28] S. Canuto, T. Salles, M. A. Gonçalves, L. Rocha, G. Ramos, L. Gonçalves, T. Rosa, and W. Martins. On efficient meta-level features for effective text classification. In *ACM International Conference on Information and Knowledge Management (CIKM 2014)*, 2014. (Cited on pages 1 and 24.)

[29] K. M. A. Chai, H. L. Chieu, and H. T. Ng. Bayesian online classifiers for text classification and filtering. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 97–104. ACM, 2002. (Cited on page 1.)

[30] D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 740–750, 2014. (Cited on page 71.)

[31] J. Cheng, L. Dong, and M. Lapata. Long short-term memory-networks for machine reading. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2016)*, 2016. (Cited on pages 2, 25, and 97.)

[32] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014. (Cited on pages 15, 89, and 100.)

[33] J. Chung, K. Cho, and Y. Bengio. A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the The 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, 2016. (Cited on page 26.)

[34] T. A. Coelho, P. P. Calado, L. V. Souza, B. Ribeiro-Neto, and R. Muntz. Image retrieval using multiple evidence ranking. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(4):408–417, 2004. (Cited on pages 3 and 53.)

[35] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pages 160–167, 2008. (Cited on pages 11, 54, and 71.)

[36] O. de Rooij, T. Kenter, and M. de Rijke. xTAS and ThemeStreams. In *Proceedings of the 13th Dutch-Belgian Workshop on Information Retrieval (DIR 13)*, 2013. (Cited on page 9.)

[37] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990. (Cited on pages 3 and 111.)

[38] B. Dolan, C. Quirk, and C. Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *COLING 2004*, 2004. (Cited on pages 22, 58, and 61.)

[39] X. L. Dong, K. Murphy, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, T. Strohmann, S. Sun, and W. Zhang. Knowledge Vault: A web-scale approach to probabilistic knowledge fusion. In *KDD 2014*, 2014. (Cited on pages 97 and 112.)

[40] S. T. Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1): 188–230, 2004. (Cited on pages 3, 12, and 111.)

[41] C. Eickhoff, J. Teevan, R. W. White, and S. T. Dumais. Lessons from the journey: a query log analysis of within-session learning. In *Proceedings of the Seventh ACM WSDM Conference (WSDM 2014)*,

2014. (Cited on page 85.)
[42] A. Fader, L. S. Zettlemoyer, and O. Etzioni. Paraphrase-driven learning for open question answering. In *Proceedings of the The 51th Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, 2013. (Cited on pages 1, 25, and 97.)
[43] M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith. Retrofitting word vectors to semantic lexicons. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL 2014)*, 2014. (Cited on page 71.)
[44] S. Fernando and M. Stevenson. A semantic similarity approach to paraphrase detection. In *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*, pages 45–52, 2008. (Cited on pages 2, 23, 53, 62, 63, and 65.)
[45] R. Ferreira, R. D. Lins, F. Freitas, S. J. Simske, and M. Riss. A new sentence similarity assessment measure based on a three-layer sentence representation. In *DocEng 2014*, 2014. (Cited on pages 2, 23, 53, and 62.)
[46] J. R. Firth. *Papers in Linguistics 1934-1951*. Oxford University Press, 1957. (Cited on pages 2 and 12.)
[47] J. R. Frank, S. J. Bauer, M. Kleiman-Weiner, D. A. Roberts, N. Tripuraneni, C. Zhang, C. Ré, E. Voorhees, and I. Soboroff. Evaluating stream filtering for entity profile updates for TREC 2013. In *TREC 2013 Working Notes*. NIST, 2013. (Cited on page 20.)
[48] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007. (Cited on page 4.)
[49] S. J. Gershman and J. B. Tenenbaum. Phrase similarity in humans and machines. In *Proceedings of the 37th Annual Conference of the Cognitive Science Society*, pages 776–781, 2015. (Cited on page 71.)
[50] D. Gillick, C. Brunk, O. Vinyals, and A. Subramanya. Multilingual language processing from bytes. In *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2016)*, 2016. (Cited on page 26.)
[51] A. Gohr, A. Hinneburg, R. Schult, and M. Spiliopoulou. Topic evolution in a stream of documents. In *Proceedings of the 2009 SIAM International Conference on Data Mining (SDM09)*, pages 859–870. SIAM, 2009. (Cited on pages 19, 20, and 33.)
[52] S. Gouws, Y. Bengio, and G. Corrado. BilBOWA: Fast bilingual distributed representations without word alignments. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, 2015. (Cited on page 113.)
[53] D. Graus, T. Kenter, M. Bron, E. Meij, and M. de Rijke. Context-based entity linking–University of Amsterdam at TAC 2012. *Text Analyis Conference (TAC 2012)*, 2012. (Cited on page 9.)
[54] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538:471–476, 2016. (Cited on pages 5 and 90.)
[55] J. Guldi. The history of walking and the digital turn: Stride and lounge in london, 1808–1851. *The Journal of Modern History*, 2012. (Cited on pages 1 and 32.)
[56] K. Gulordava and M. Baroni. A distributional similarity approach to the detection of semantic change in the Google Books Ngram corpus. In *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*, 2011. (Cited on pages 21, 31, and 39.)
[57] D. Hall, D. Jurafsky, and C. D. Manning. Studying the history of ideas using topic models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2008)*, 2008. (Cited on pages 1, 19, 20, 21, and 32.)
[58] L. Han, A. Kashyap, T. Finin, J. Mayfield, and J. Weese. UMBC EBIQUITY-CORE: Semantic textual similarity systems. In *Second Joint Conference on Lexical and Computational Semantics (*SEM-2013)*, 2013. (Cited on pages 23, 53, and 57.)
[59] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954. (Cited on pages 2 and 12.)
[60] Q. He, B. Chen, J. Pei, B. Qiu, P. Mitra, and L. Giles. Detecting topic evolution in scientific literature: how can citations help? In *ACM International Conference on Information and Knowledge Management (CIKM 2009)*, 2009. (Cited on pages 1 and 19.)
[61] M. Henaff, J. Weston, A. Szlam, A. Bordes, and Y. LeCun. Tracking the world state with recurrent entity networks. *Proceedings of the International Conference on Learning Representations (ICLR 2017)*, 2017. (Cited on pages 89 and 90.)
[62] K. M. Hermann and P. Blunsom. Multilingual distributed representations without word alignment. In *Proceedings of the International Conference on Learning Representations (ICLR 2014)*, 2014. (Cited on page 24.)

[63] K. M. Hermann and P. Blunsom. Multilingual models for compositional distributed semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, pages 58–68, 2014. (Cited on page 24.)

[64] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, 2015. (Cited on pages 2, 5, 16, 25, 26, 85, 97, 98, 100, and 101.)

[65] D. Hewlett, A. Lacoste, L. Jones, I. Polosukhin, A. Fandrianto, J. Han, M. Kelcey, and D. Berthelot. WIKIREADING: A novel large-scale language understanding task over Wikipedia. In *Proceedings of the The 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, 2016. (Cited on pages 1, 2, 5, 25, 85, 97, 99, 104, 105, and 106.)

[66] G. Heyer, F. Holz, and S. Teresniak. Change of topics over time-tracking topics by their change of meaning. In *Proceedings of The International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KDIR 2009)*, volume 9, pages 223–228, 2009. (Cited on pages 1, 21, 31, and 42.)

[67] F. Hill, K. Cho, and A. Korhonen. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL 2016)*, 2016. (Cited on pages 25, 71, and 77.)

[68] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. (Cited on page 15.)

[69] F. Holz and S. Teresniak. Towards automatic detection and tracking of topic change. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 327–339. Springer, 2010. (Cited on pages 21, 31, and 42.)

[70] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 2042–2050, 2014. (Cited on pages 1, 22, 24, 55, 61, 62, 63, 66, and 71.)

[71] P. Huijnen, F. Laan, M. de Rijke, and T. Pieters. A digital humanities approach to the history of science. In *Workshops at the International Conference on Social Informatics*, pages 71–85. Springer, 2013. (Cited on pages 1 and 32.)

[72] A. Islam and D. Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2008. (Cited on pages 2, 3, 22, 53, 61, 62, and 63.)

[73] Y. Ji and J. Eisenstein. Discriminative improvements to distributional sentence similarity. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, pages 891–896, 2013. (Cited on pages 22 and 64.)

[74] J. J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the 10th International Conference on Research in Computational Linguistics, ROCLING97*, 1997. (Cited on page 2.)

[75] V. Jijkoun and M. de Rijke. Recognizing textual entailment using lexical similarity. In *Proceedings Pascal 2005 Textual Entailment Challenge Workshop*, 2005. (Cited on pages 3 and 53.)

[76] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016. (Cited on pages 26, 98, and 102.)

[77] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, 2013. (Cited on page 100.)

[78] T. Kenter. Filtering documents over time for evolving topics - the University of Amsterdam at TREC 2013 KBA CCR. In *Proceedings of the Twenty-Second Text REtrieval Conference (TREC 2013)*, 2013. (Cited on pages 9 and 20.)

[79] T. Kenter and M. de Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM 2015)*, pages 1411–1420, 2015. (Cited on pages 1, 2, 8, 24, and 71.)

[80] T. Kenter and M. de Rijke. Attentive memory networks: Efficient machine reading for conversational search. In *Proceedings of the The First International Workshop on Conversational Approaches to Information Retrieval (CAIR'17)*, 2017. (Cited on pages 8, 26, 98, and 102.)

[81] T. Kenter, D. Graus, E. Meij, and M. de Rijke. Time-aware chi-squared for document filtering over time. In *SIGIR 2013 Workshop on Time-aware Information Access (TAIA 2013)*, 2013. (Cited on page 9.)

[82] T. Kenter, K. Balog, and M. de Rijke. Evaluating document filtering systems over time. *Information Processing & Management*, 51(6):791–808, 2015. (Cited on pages 8, 49, and 112.)

[83] T. Kenter, M. Wevers, P. Huijnen, and M. de Rijke. Ad hoc monitoring of vocabulary shifts over time. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM 2015)*, pages 1191–1200, 2015. (Cited on pages 1, 8, and 71.)

[84] T. Kenter, A. Borisov, and M. de Rijke. Siamese CBOW: Optimizing word embeddings for sentence representations. In *Proceedings of the The 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, pages 941–951, 2016. (Cited on pages 1, 8, and 97.)

[85] T. Kenter, A. Borisov, C. Van Gysel, M. Dehghani, M. de Rijke, and B. Mitra. Neural networks for information retrieval. In *arXiv preprint arXiv:1707.04242*, 2017. (Cited on page 8.)

[86] T. Kenter, L. Jones, and D. Hewlett. Byte-level machine reading across morphologically varied languages. In *under review*, 2017. (Cited on pages 2 and 8.)

[87] S. Kim, L. F. D'Haro, R. E. Banchs, J. D. Williams, and M. Henderson. The fourth dialog state tracking challenge. In *Dialogues with Social Robots*, pages 435–449. Springer, 2017. (Cited on page 85.)

[88] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014. (Cited on pages 1, 22, and 59.)

[89] Y. Kim, I. Yi-Chiu., K. Hanaki, D. Hegde, and S. Petrov. Temporal analysis of language through neural language models. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, pages 61–65, 2014. (Cited on pages 1, 20, 21, 31, and 71.)

[90] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)*, 2016. (Cited on pages 26 and 98.)

[91] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR 2015)*, 2015. (Cited on pages 89 and 104.)

[92] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-Thought vectors. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pages 3294–3302. Curran Associates, Inc., 2015. (Cited on pages 24, 71, 74, 75, and 80.)

[93] J. Kiseleva and M. de Rijke. Evaluating personal assistants on mobile devices. In *1st International Workshop on Conversational Approaches to Information Retrieval (CAIR'17)*. ACM, August 2017. (Cited on page 85.)

[94] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003. (Cited on page 1.)

[95] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007. (Cited on page 1.)

[96] M. Koolen, F. Adriaans, J. Kamps, and M. de Rijke. A cross-language approach to historic document retrieval. In *Proceedings of the European Conference on Information Retrieval (ECIR 2006)*, 2006. (Cited on page 114.)

[97] H. Kozima. Similarity between words computed by spreading activation on an english dictionary. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics (EACL 1993)*, 1993. (Cited on page 2.)

[98] V. Kulkarni, R. Al-Rfou, B. Perozzi, and S. Skiena. Statistically significant detection of linguistic change. In *Proceedings of the 23rd International World Wide Web Conference (WWW2014)*, 2014. (Cited on pages 21 and 31.)

[99] A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, V. Zhong, R. Paulus, and R. Socher. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)*, 2016. (Cited on pages 5, 25, 26, 88, 89, and 91.)

[100] M. Kusner, Y. Sun, N. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, pages 957–966, 2015. (Cited on pages 75 and 110.)

[101] J. Kuukkanen. Making sense of conceptual change. *History and theory*, 2008. (Cited on pages 3, 20, and 34.)

[102] S. Lauly, H. Larochelle, M. Khapra, B. Ravindran, V. C. Raykar, and A. Saha. An autoencoder approach to learning bilingual word representations. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 1853–1861, 2014. (Cited on page 24.)

[103] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning*, 2014. (Cited on pages 22 and 54.)

[104] M. D. Lee, B. Pincombe, and M. B. Welsh. An empirical evaluation of models of text document similarity. *Cognitive Science*, 2005. (Cited on page 110.)

[105] H. Li and J. Xu. Semantic matching in search. *Foundations and Trends in Information Retrieval*, 7(5): 343–469, 2014. (Cited on page 53.)

[106] Y. Li, D. McLean, Z. A. Bandar, J. D. O'shea, and K. Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 18(8): 1138–1150, 2006. (Cited on pages 23 and 62.)

[107] W. Ling, T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*, 2015. (Cited on pages 26, 98, and 101.)

[108] M. C. Lintean and V. Rus. Measuring semantic similarity in short texts through greedy pairing and word semantics. In *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference (FLAIRS Conference 2012)*, 2012. (Cited on page 23.)

[109] M.-T. Luong and C. D. Manning. Achieving open vocabulary neural machine translation with hybrid word-character models. In *Proceedings of the The 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, 2016. (Cited on pages 17, 98, and 101.)

[110] T. Luong, R. Socher, and C. D. Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning (CoNLL-2013)*, 2013. (Cited on page 98.)

[111] A. Lynum, P. Pakray, B. Gambäck, and S. Jimenez. NTNU: Measuring semantic similarity with sublexical feature representations and soft cardinality. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 448–453, 2014. (Cited on page 77.)

[112] I. Mani and M. T. Maybury. *Advances in automatic text summarization*, volume 293. MIT Press, 1999. (Cited on page 1.)

[113] E. Marsi, H. Moen, L. Bungum, G. Sizov, B. Gambäck, and A. Lynum. NTNU-CORE: Combining strong features for semantic similarity. *Second Joint Conference on Lexical and Computational Semantics (*SEM-2013)*, 2013. (Cited on pages 3, 23, and 57.)

[114] C. Martinez-Ortiz, T. Kenter, M. Wevers, P. Huijnen, J. Verheul, and J. van Eijnatten. Design and implementation of ShiCo: Visualising shifting concepts over time. In *Proceedings of the 3rd International Workshop on Computational History (HistoInformatics 2016)*, 2016. (Cited on pages 1 and 8.)

[115] C. Martinez-Ortiz, T. Kenter, M. Wevers, P. Huijnen, J. Verheul, and J. van Eijnatten. ShiCo: A visualization tool for shifting concepts through time. In *Proceedings of the 3rd DH Benelux Conference (DH Benelux 2016)*, 2016. (Cited on pages 1 and 9.)

[116] D. Metzler, S. Dumais, and C. Meek. Similarity measures for short segments of text. In *Proceedings of the European Conference on Information Retrieval (ECIR 2007)*, 2007. (Cited on pages 2 and 53.)

[117] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of Twenty-First AAAI Conference on Artificial Intelligence (AAAI 2006)*, 2006. (Cited on pages 2, 23, 53, and 62.)

[118] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, 2013. (Cited on pages 1, 2, 3, 11, 12, 21, 24, 34, 46, 54, 71, and 111.)

[119] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 3111–3119, 2013. (Cited on pages 1, 2, 3, 11, 12, 21, 54, 60, 71, 75, and 111.)

[120] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston. Key-value memory networks for directly reading documents. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2016)*, 2016. (Cited on pages 5, 25, 98, 102, and 114.)

[121] G. A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995. (Cited on pages 2 and 111.)

[122] F. Moretti and D. Pestre. Bankspeak. *New Left Review*, 2015. (Cited on page 32.)

[123] D. Odijk, E. Meij, D. Graus, and T. Kenter. Multilingual semantic linking for video streams: Making "ideas worth sharing" more accessible. In *Proceedings of the 2nd International Workshop on Web of Linked Entities at WWW2013 (WoLE2013)*, 2013. (Cited on page 9.)

[124] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In

Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 2014. (Cited on pages 2, 11, 12, 21, 54, 60, and 111.)

[125] O. Press and L. Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017)*, 2017. (Cited on page 103.)

[126] C. Quirk, C. Brockett, and W. B. Dolan. Monolingual machine translation for paraphrase generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*, 2004. (Cited on pages 22, 58, and 61.)

[127] F. Radlinski and N. Craswell. A theoretical framework for conversational search. In *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval (CHIIR 2017)*, pages 117–126. ACM, 2017. (Cited on page 85.)

[128] R. Reinanda, E. Meij, and M. de Rijke. Mining, ranking and recommending entity aspects. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2015)*, pages 263–272, 2015. (Cited on page 71.)

[129] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. (Cited on page 56.)

[130] T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, and P. Blunsom. Reasoning about entailment with neural attention. In *Proceedings of the International Conference on Learning Representations (ICLR 2015)*, 2015. (Cited on page 97.)

[131] A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, 2015. (Cited on pages 1 and 5.)

[132] A. Saha and V. Sindhwani. Learning evolving and emerging topics in social media: a dynamic nmf approach with temporal regularization. In *Proceedings of the Fifth ACM WSDM Conference (WSDM 2012)*, 2012. (Cited on page 20.)

[133] G. Salton, A. Wong, and C.-S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. (Cited on page 3.)

[134] F. Šarić, G. Glavaš, M. Karan, J. Šnajder, and B. D. Bašić. TakeLab: Systems for measuring semantic text similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM-2013)*, 2012. (Cited on pages 2, 3, 23, and 53.)

[135] I. V. Serban, A. Sordoni, R. Lowe, L. Charlin, J. Pineau, A. Courville, and Y. Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017. (Cited on pages 5, 25, and 85.)

[136] A. Severyn and A. Moschitti. 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 (SIGIR 2015)*, pages 373–382, 2015. (Cited on page 24.)

[137] L. Shang, Z. Lu, and H. Li. Neural responding machine for short-text conversation. In *Proceedings of the The 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, 2015. (Cited on page 16.)

[138] P. Shrestha. Corpus-based methods for short text similarity. *Rencontre des Étudiants Chercheurs en Informatique pour le Traitement automatique des Langues*, 2011. (Cited on pages 22, 61, 62, and 63.)

[139] R. Socher, E. H. Huang, J. Pennin, C. D. Manning, and A. Y. Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, 2011. (Cited on pages 22, 53, 54, and 64.)

[140] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012)*, pages 1201–1211, 2012. (Cited on pages 4 and 71.)

[141] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, 2013. (Cited on pages 55 and 66.)

[142] A. Sordoni, Y. Bengio, H. Vahabi, C. Lioma, J. Grue Simonsen, and J.-Y. Nie. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *ACM International Conference on Information and Knowledge Management (CIKM 2015)*, 2015. (Cited on pages 3, 16, and 25.)

[143] K. Sparck Jones and van Rijsbergen C. Report on the need for and provision of an "ideal" information retrieval test collection. *British Library Research and Development Report 5266*, 1975. (Cited on page 40.)

[144] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-end memory networks. In *Advances in*

Neural Information Processing Systems 28 (NIPS 2015), 2015. (Cited on pages 5, 25, 26, 86, 90, 91, 98, and 102.)

[145] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014. (Cited on pages 1, 5, 14, 16, and 100.)

[146] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, 2016. (Cited on page 75.)

[147] K. Tran, A. Bisazza, and C. Monz. Recurrent memory networks for language modeling. In *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2016)*, 2016. (Cited on page 25.)

[148] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. (Cited on page 1.)

[149] O. Vinyals and Q. Le. A neural conversational model. *ICML Deep Learning Workshop 2015*, 2015. (Cited on page 16.)

[150] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, 2015. (Cited on page 17.)

[151] N. Voskarides, E. Meij, M. Tsagkias, M. de Rijke, and W. Weerkamp. Learning to explain entity relationships in knowledge graphs. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and The 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2015)*, pages 564–574, 2015. (Cited on page 71.)

[152] D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014. (Cited on page 99.)

[153] S. Wang and J. Jiang. Machine comprehension using match-LSTM and answer pointer. In *Proceedings of the International Conference on Learning Representations (ICLR 2017)*, 2017. (Cited on page 97.)

[154] S. Wang, S. Schlobach, and M. Klein. Concept drift and how to identify it. *Web Semantics*, 2011. (Cited on pages 1, 20, 31, 32, and 34.)

[155] X. Wang and A. McCallum. Topics over time: a non-markov continuous-time model of topical trends. In *Proceedings of the Twelfth ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD 2006)*, 2006. (Cited on pages 19 and 33.)

[156] Z. Wang, K. Zhao, H. Wang, X. Meng, and J.-R. Wen. Query understanding through knowledge-based conceptualization. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15)*, 2015. (Cited on page 85.)

[157] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang. Clustering user queries of a search engine. In *Proceedings of the 10th international conference on World Wide Web*, pages 162–168. acm, 2001. (Cited on page 3.)

[158] J. Weston, S. Chopra, and A. Bordes. Memory networks. In *Proceedings of the International Conference on Learning Representations (ICLR 2015)*, 2015. (Cited on pages 5, 25, 26, 86, 98, and 102.)

[159] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov. Towards AI-complete question answering: A set of prerequisite toy tasks. In *Proceedings of the International Conference on Learning Representations (ICLR 2016)*, 2016. (Cited on pages 1, 85, 88, and 89.)

[160] M. Wevers, T. Kenter, and P. Huijnen. Concepts through time: Tracing concepts in dutch newspaper discourse (1890-1990) using word embeddings. In *Digital Humanities 2015 (DH 2015)*, 2015. (Cited on pages 1 and 9.)

[161] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu. Towards universal paraphrastic sentence embeddings. *Proceedings of the International Conference on Learning Representations (ICLR 2016)*, 2016. (Cited on page 24.)

[162] D. T. Wijaya and R. Yeniterzi. Understanding semantic change of words over centuries. In *Proceedings of the 2011 International Workshop on Detecting and Exploiting Cultural Diversity on the Social Web (DETECT'11)*, pages 35–40, 2011. (Cited on pages 21 and 31.)

[163] Y. Xiao and K. Cho. Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*, 2016. (Cited on pages 1, 22, 98, 102, 104, and 110.)

[164] C. Xiong, S. Merity, and R. Socher. Dynamic memory networks for visual and textual question answering. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)*, 2016. (Cited on pages 1, 3, 5, 16, 25, 26, 87, 88, 89, and 90.)

[165] Y. Yang, W. tau Yih, and C. Meek. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*, 2015. (Cited on pages 1, 25, and 99.)

[166] Z. Yang, Y. Yuan, Y. Wu, R. Salakhutdinov, and W. W. Cohen. Encode, review, and decode: Reviewer module for caption generation. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, 2016. (Cited on pages 26, 27, 98, and 103.)

[167] W.-t. Yih, K. Toutanova, J. C. Platt, and C. Meek. Learning discriminative projections for text similarity measures. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 247–256, 2011. (Cited on page 24.)

[168] W. Yin and H. Schütze. Convolutional neural network for paraphrase identification. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL 2015)*, pages 901–911, 2015. (Cited on page 24.)

[169] L. Yu, K. M. Hermann, P. Blunsom, and S. Pulman. Deep learning for answer sentence selection. In *NIPS 2014 Deep Learning and Representation Learning Workshop*, 2014. (Cited on page 71.)

[170] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pages 649–657, 2015. (Cited on pages 1, 26, and 98.)

[171] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 19–27, 2015. (Cited on page 74.)

[172] W. Y. Zou, R. Socher, D. M. Cer, and C. D. Manning. Bilingual word embeddings for phrase-based machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, pages 1393–1398, 2013. (Cited on page 71.)

Samenvatting

Eén van de heilige gralen binnen het onderzoek naar computerprogramma's die met mensen communiceren in natuurlijke taal is het doorstaan van de Turing test. De computer zou zo natuurlijk moeten communiceren dat het voor mensen onmogelijk is om te weten of ze met een computer of een ander mens in gesprek zijn. Het onderzoeksgebied van *natural language understanding* — dat bestudeert hoe computers teksten in natuurlijke taal kunnen begrijpen — speelt een centrale rol in dit onderzoek. *Natural language understanding* kan op verschillende niveaus worden bestudeerd. In dit proefschrift wordt een bijdrage gedaan aan het onderzoek naar het automatisch begrijpen van natuurlijke taal op het niveau van woorden, korte teksten (zoals zinnen) en volledige documenten.

Taal begrijpen op woordniveau betekent begrijpen hoe betekenissen van woorden zich tot elkaar verhouden. Bijvoorbeeld, betekenen twee woorden hetzelfde? Heeft een woord dezelfde betekenis die het, zeg, 50 jaar geleden had? Of, zoals de centrale vraag in het eerste deel van dit proefschrift is: kunnen we er op een automatische manier achter komen welke woorden mensen door de tijd heen gebruiken om een bepaald concept beschrijven?

Wanneer de betekenis van woorden goed gevangen kan worden, kunnen we proberen om langere stukken tekst, zoals zinnen, te gaan begrijpen. De vraag die we stellen in het tweede deel van dit proefschrift is: kunnen we, door de betekenissen van woorden in twee zinnen met elkaar te vergelijken, bepalen of de zinnen hetzelfde betekenen?

Het derde en laatste deel van dit proefschrift gaat over tekstbegrip op documentniveau. De taak die we bestuderen kan omschreven worden als "begrijpend lezen voor computers," waarbij de computer een document leest en er vragen over beantwoordt.

Het tekstbegrip van computers is op dit moment beter dan ooit. Een van de toepassingen waarin dit tot uitdrukking komt is dialoogsystemen. Voor het eerst in de geschiedenis kunnen mensen op enigszins redelijk niveau communiceren met zogenaamde *digital assistants* zoals de Google Assistant, Amazons Alexa, Siri van Apple en Microsofts Cortana. We zijn echter nog ver verwijderd van alwetende systemen als de Star Trek computer of andere hyperintelligente kunstmatige intelligenties die in science fiction boeken en films voorkomen, en de huidige systemen zullen waarschijnlijk niet binnenkort de Turing test doorstaan. Waar ze wel goed in zijn is het helpen in allerlei alledaagse taken, zoals wekkers zetten, afspraken maken in elektronische agenda's, het licht aandoen, een televisieprogramma aanzetten en weersvoorspellingen geven. En dit alles simpelweg doordat het ze gevraagd wordt in natuurlijke taal. Nu er meer en meer functionaliteiten worden toegevoegd aan dit soort systemen zal de tijd uitwijzen welke daarvan aanslaan. Dit proces zelf heeft op zijn beurt weer invloed op de manier waarop we met computers zullen communiceren. Zullen het handige assistenten blijven die alledaagse taken voor ons kunnen verrichten? Of zullen we uiteindelijk een uitgebreide conversatie met ze kunnen voeren, zoals die waar de kunstmatige wezens op de omslag in verwikkeld lijken te zijn?

Summary

A long-standing challenge for computers communicating with humans is to pass the Turing test, i.e., to communicate in such a way that it is impossible for humans to determine whether they are talking to a computer or another human being. The field of *natural language understanding* — which studies automatic means of capturing the semantics of textual content — plays a central part in this long-term goal of artificial intelligence research. Natural language understanding can itself be understood at different levels. In this thesis, we make contributions to automatic understanding of text at the level of words, short texts, and full documents.

Understanding texts at the word level, means understanding how words relate to each other semantically. For example, do two words or phrases mean approximately the same thing? Does a particular word still mean the same thing it used to, say, 50 years ago? Or, as is the question central to the first part of this thesis, can we automatically detect which words people used in different periods in time to refer to a particular concept?

When word-level semantics are understood to a sufficient degree, an attempt can be made at capturing the meaning of short pieces of text, such as sentences. The question we ask ourselves in the second part of this thesis is: can we automatically determine, from the word-level up, if two sentences have a similar meaning?

Finally, in the third part of this thesis, document-level text understanding is the focus of our interest. In particular, we study multiple approaches to the reading comprehension task, where a computer reads a document and answers questions about it.

Today, text understanding is at an unprecedented level. It plays a role in many tasks, among which are dialogue systems, or conversational agents. For the first time in history people can have more or less meaningful conversations with digital assistants, like the Google Assistant, Amazon's Alexa, Apple's Siri and Microsoft's Cortana. Still, the conversational systems we have today are miles away from omniscient systems like the Star Trek computer or any other hyper-intelligent AI envisioned in science fiction, and they will probably not pass the Turing test any time soon. However, they can perform mundane tasks like setting alarms, making appointments in a personal calendar, switching on the lights, playing television shows, or reporting what the weather is going to be like, all simply by being asked to do so in natural language. As new functionalities are being added, time will tell which ones will catch on, a process itself influencing how we will interact with computers. Do we stick to them being personal assistants that can do simple tasks for us? Or do we end up having full-fledged conversations with them, like the one the bots on the cover of this book seem to be engaged in?

About the cover

I took the photo on the front cover at the Pudong International Airport in Shanghai. The research in this thesis is about understanding texts by automatic means, and is meant to contribute to the larger field of artificial intelligence, of which text understanding is but a small part. The figures, to me, symbolize artificial but humanoid beings. One could imagine research in artificial intelligence, years from now, culminating in beings like this. Beings who behave very naturally, like us, but who, at the same time, are very different from us.

The picture on the back I took while on a mountain hike in Bolivia. It depicts a pile of rocks on a mountain track, put there one by one by people passing by. The picture, to me, symbolizes the journey of doing a PhD. Much as on the hike I took the picture on, the further you get, the less people you see (that is, the less people are involved in the same research you are). You find your own route and pick your own trails, supported by an experienced guide, familiar with the area. Few people ever go exactly where you are going, and only some of those who do mark their presence with a stone (i.e., a publication). However little the contributions may seem, progress is always being made. Every stone contributes to the pile being different from what it ever was before.

Text Understanding for Computers

A long-standing challenge for computers communicating with humans is to pass the Turing test, i.e., to communicate in such a way that it is impossible for humans to determine if they are talking to a computer or another human being. The field of *natural language understanding* — which studies automatic means of capturing the semantics of textual content — plays a central part in this long-term goal of artificial intelligence research. Natural language understanding can itself be understood at different levels. In this thesis, we make contributions to automatic understanding of text at the level of words, short texts, and full documents.