UNIVERSITY OF CAPE TOWN

Natural Language Processing on Data Warehouses

Author Stiaan Maree

Supervisor Prof. Ian Durbach

A thesis submitted in partial fulfilment of the requirements for the degree of
Masters in Data Science in the Faculty of Science
Department of Statistical Sciences
December 2018

Abstract

The main problem addressed in this research was to use natural language to query data in a data warehouse. To this effect, two natural language processing models were developed and compared on a classic star-schema sales data warehouse with sales facts and date, location and item dimensions. Utterances are queries that people make with natural language, for example, "What is the sales value for mountain bikes in Georgia for 1 July 2005?" The first model, the heuristics model, implemented an algorithm that steps through the sequence of utterance words and matches the longest number of consecutive words at the highest grain of the hierarchy. In contrast, the embedding model implemented the word2vec algorithm to create different kinds of vectors from the data warehouse. These vectors are aggregated and then the cosine similarity between vectors was used to identify concepts in the utterances that can be converted to a programming language. To understand question style, a survey was set up which then helped shape random utterances created to use for the evaluation of both methods.

The first key insight and main premise for the embedding model to work is a three-step process of creating three types of vectors. The first step is to train vectors (word vectors) for each individual word in the data warehouse; this is called word embeddings. For instance, the word 'bike' will have a vector. The next step is when the word vectors are averaged for each unique column value (column vectors) in the data warehouse, thus leaving an entry like 'mountain bike' with one vector which is the average of the vectors for 'mountain' and 'bike'. Lastly, the utterance by the user is averaged (utterance vectors) by using the word vectors created in step one, and then, by using cosine similarity, the utterance vector is matched to the closest column vectors in order to identify data warehouse concepts in the utterance.

The second key insight was to train word vectors firstly for location, then separately for item - in other words, per dimension (one set for location, and one set for item). Removing stop words was the third key insight, and the last key insight was to use Global Vectors to instantiate the training of the word vectors.

The results of the evaluation of the models indicated that the embedding model was ten times faster than the heuristics model. In terms of accuracy, the embedding algorithm (95.6% accurate) also outperformed the heuristics model (70.1% accurate).

The practical application of the research is that these models can be used as a component in a chatbot on data warehouses. Combined with a Structured Query Language query generation component, and building Application Programming Interfaces on top of it, this facilitates the quick and easy distribution of data; no knowledge of a programming language such as Structured Query Language is needed to query the data.

Keywords: Natural language processing, data warehouses, embedding

Acknowledgements

Firstly, I would sincerely like to thank Prof. Ian Durbach, my supervisor, who provided guidance and support throughout the project. Thank you for keeping an eye on my progress, for always being available and for directing my understanding of the topic.

To my mom, dad, brother, mother-in-law and father-in-law, thank you for always supporting me in endeavours of learning something new.

Lastly, thank you to my wife and children, for sacrificing so much time for this project to see the light of day.

Plagiarism Declaration

I know the meaning of plagiarism and declare that all of the work in the document, save for that which is properly acknowledged, is my own.

Signature:

Signed by candidate

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface
GloVe	Global Vectors for Word Representation
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NoSQL	Not Only SQL
POS	Part-of-Speech
RNN	Recurrent Neural Network
SQL	Structured Query Language
TF-IDF	Term Frequency Inverse Document Frequency
TreeNN	Recursive Neural Network

Chapter 1

Introduction

1.1 Background

The Turing Test was designed to assess a computer's operational intelligence (Turing, 1950). The machine would pass the test if a human could not tell if the responses of an interaction came from the machine or another human. Natural language processing (NLP) is at the heart of such a system, along with knowledge representation, automated reasoning and machine learning. Natural language processing can be defined as a research area exploring how computers can be used to understand and use natural language text or speech (Chowdhury, 2003).

Data volume is increasing faster than processing power (Jagadish et al., 2014), surpassing Moore's Law (Loukides, 2010), which states that the density of components per integrated circuit doubles every two years. In other words, the data to be processed grows faster than the ability to process it. The term 'Big Data' has been misused in industry as it has been hailed as a silver bullet, where the actual strength hidden in big data is seldom used. The rise of big data led to the establishment of data science as an industry. The industry is currently experiencing a mad rush for big data experts and data scientists, which is caused by user-generated content on the web and the need for information aggregation, which has grown exponentially (Cambria and White, 2014).

Where natural language processing fits into data science is a point of contention amongst researchers. Natural language processing can be seen as part of data conditioning, the first step of any data analysis project (Loukides, 2010), and some researchers classify it under the heading of text analytics (Gandomi and Haider, 2015). In contrast, other researchers see natural language processing as more than just a stepping stone for other analysis and distinguish between natural language processing and text analysis, with the main difference being in the problem that each tries to solve (Cady, 2017). Natural language processing analyses user utterances to understand the user's intent. Text analysis, in contrast, uses techniques to extract information from text data and then classify, group and cluster (Cady, 2017).

The two main reasons for pursuing natural language processing are for machines to communicate with humans and for machines to gain information from written language (Russel and Norvig, 2010). This research is concerned with the latter. In particular, this research focuses on understanding user generated utterances to query data. Another motivation for natural language processing is computational linguistics, where the goal is to use text data to understand human language and the mechanisms which it uses. Computational linguistics has a scientific rather than engineering motivation (Jurafsky and Martin, 2018).

Table 1.1: Examples of utterances that has been transformed to SQL queries, with each query extracting all the relevant information from the utterance.

Utterance 1	"What is the sales value for mountain bikes in Georgia for 1 July 2005?"
SQL 1	SELECT SUM(value) FROM sales WHERE province LIKE "Georgia" AND sub_category LIKE "mountain bikes" AND Date = CAST("2005-07-01" AS DATE)
Utterance 2	"What is the top item sold in the United States on 1 July 2005?"
SQL 2	SELECT TOP 1 product FROM sales WHERE country LIKE "United States" AND Date = CAST("2005-07-01" AS DATE) GROUP BY product ORDER BY SUM(value) DESC

1.2 Statement of the Problem

The main problem this research is trying to solve is to create a solution which enables the interaction with a data warehouse, using natural language. The solution needs to be able to transform the natural language into a Structured Query Language (SQL) statement which can be sent to a data warehouse.

Natural language processing questions asked by the user are called utterances. To illustrate the difficulty, consider the following data warehouse and utterances. The data warehouse follows a flat file implementation of a star schema data warehouse, which houses the sales value and sales quantity as facts in the middle, surrounded by the date, location and item dimensions, linked to the central fact. Both the location and the item dimensions consist of hierarchies. The location hierarchy starts with country at the highest level, comprising of several provinces. Each province has a few cities, and each city has few shops. Similarly, the item hierarchy has three levels, ranging from category through sub-category to product. The data can be seen in Table 1.2.

The questions in Table 1.1 have to be transformed from utterances to SQL queries. The finished result for the examples can be seen in Table 1.1.

Firstly, the system needs to identify 'what' the user wants returned; this is called the user intent. Next, the solution has to narrow the query down to the parameters, as mentioned by the user, by identifying entities.

To identify 'what' the user wants, the system needs to find the correct column in the data warehouse, and the part of the utterance that refers to this column is identified; for example, in utterance 1: 'sales value' from the utterance refers to the 'value' column in the data warehouse and 'item' in utterance 2 refers to the 'product' column in the data warehouse.

The identification of the terms to use to narrow the query down is the heart of the problem. The following issues needs to be addressed by the perfect solution:

- How does the system know that 'Georgia' represents a parameter to be used for narrowing the query, and not 'Georgia for'?
- How does the system know that Georgia is a province, and not a shop?
- Would the system understand that 'Long-Sleeve Logo Jersey, M' is the same as 'Logo Jersey M Long-

Table 1.2: The Adventureworks data warehouse consists of two facts, namely value and quantity, plus three dimensions, namely, date, item and location. Both the item and location dimension each consists of a hierarchy with a few columns each. The item hierarchy comprises the product, sub_category and the category column. The location hierarchy has four columns: shop, city, province and country.

value	quantity	date	product	sub_category	category	shop	city	province	country
4079.988	2	01/07/2005	Mountain-100 Silver, 44	Mountain Bikes	Bikes	Better Bike Shop	Austell	Georgia	United States
86.5212	3	01/07/2005	Long-Sleeve Logo Jersey, M	Jerseys	Clothing	Better Bike Shop	Austell	Georgia	United States
34.2	6	01/07/2005	Mountain Bike Socks, M	Socks	Clothing	Better Bike Shop	Austell	Georgia	United States
10.373	2	01/07/2005	AWC Logo Cap	Caps	Clothing	Better Bike Shop	Austell	Georgia	United States
80.746	4	01/07/2005	Sport-100 Helmet, Blue	Helmets	Accessories	Better Bike Shop	Austell	Georgia	United States
874.794	1	01/07/2005	Road-450 Red, 52	Road Bikes	Bikes	Primary Cycle Shop	Hamburg	Hamburg	Germany
809.76	1	01/07/2005	HL Mountain Frame - Black, 48	Mountain Frames	Components	Original Bicycle Supply Company	Toronto	Ontario	Canada
714.7043	1	01/07/2005	HL Mountain Frame - Black, 42	Mountain Frames	Components	Original Bicycle Supply Company	Toronto	Ontario	Canada

Sleeve'?

- The word 'Hamburg' appears in the city and province columns; to which column did the user refer?

It can be seen that the generation of SQL queries from user utterances has many levels of problems to be addressed, which this research aims to do.

1.3 Objectives

The main objective of the research is to compare two types of natural language processing models on the same sales data warehouse. This comprises four activities:

- Develop and program a heuristics model;
- Develop and program an embedding model;
- Gain insight on user question style and create random utterances for testing;
- Compare approaches in terms of accuracy and computing time.

The heuristics model uses an algorithm that tries to map utterance words to concepts in the data warehouse using the hierarchies in a top-down approach; for example, province appears above city in the location hierarchy, thus utterance words will be matched to province before city in the algorithm. At each step, as the concepts are mapped to the words, the utterance gets shorter and is sent to the next concept. This is not a parallel algorithm. The main drawback for this model is that it is computationally very heavy.

The embedding model addressed this main drawback by creating vector representations for each concept in the data warehouse and testing the full utterance against each column in the data warehouse in parallel. A key feature of the embedding model is that each concept in the data warehouse gets represented by a numerical vector (i.e., a point in multidimensional space). The vector contains information about the 'meaning' of the concept, resulting in similar concepts being close to one another in multi-dimensional space. These vectors can then be used to do a match by averaging vectors over multiple words in an utterance (which gives another vector) and seeing which concept in the data warehouse is closest.

A survey was set up to understand how users would ask questions from a specific data warehouse. The survey was run to understand the question style. The results from the survey were used to create random utterances for testing. The models were then compared against the randomly generated user utterances for accuracy and computing time.

1.4 Practical Applications

The practical applications for a natural language-processing solution on data warehouses are apparent, in the ubiquitous distribution of data within the organisation. SQL-abled persons are no longer gatekeepers to the distribution of data. A chatbot is a humanlike conversational entity, which, in essence, is a program that conducts conversations though voice or text methods (Shaikh et al., 2016). Chatbots can be deployed on top of data warehouses using the natural language systems, as developed by this research. Current machine learning-based language understanding solutions, such as Microsoft's Luis.ai, Amazon's Lex or Facebook's

Wit.ai, require the user to supply examples of utterances upfront, for the system to have training data to train language models.

Both solutions proposed in this research do not have this limitation; the solutions can be deployed on top of the data warehouse and can be used without requiring further input. The heuristics model, however, is computationally heavy, and the development of an embedding model would present a breakthrough in terms of out-of-the-box readiness combined with speed of query processing.

1.5 Limitations

The source dataset presents a restriction in the schema of the data. Sales data was used for the research, as would be found in a data warehouse. The main difference between a data warehouse and a database is normalisation of the data. A database mostly uses third form normalisation, whereas a data warehouse is denormalised. In a data warehouse, the facts (for example, sales quantity or sales value) are in the centre of the schema, joined with dimensions tables (for example, item, location or date). However, in this research, the facts and dimensions were joined into one table, thus creating the limitation that table joins are not navigated by either model. Future research can focus on improving these limitations.

Chapter 2

Literature review

The literature review starts by looking at the human and language. Why do humans have language, and how do we make sense of it? The subsequent section narrows down the domain of language by looking at how machines do statistical natural language processing. In other words, how do machines make sense of human language? The scope of the literature review is narrowed again in the following section when looking at deep learning-based methods for natural language processing. Next, the domain on which this research focuses natural language is discussed, namely, databases and specifically data warehouses: looking at the problem of organising data, and how databases and data warehouses play a role in storing our ability to manage data. Lastly, the very narrow topic of natural language processing on databases are covered.

2.1 Language

"Language is entwined with human life. We use it to inform and persuade, but also to threaten, to seduce, and of course to swear. It reflects the way we grasp reality, and also the image of ourselves we try to project to others, and the bonds that tie us to them. It is, I hope to convince you, a window into human nature" (Pinker, 2007).

2.1.1 Language and the Brain

Neurolinguistics is the study of how the human brain enables language. Many types of researchers contribute to the interdisciplinary field, for example, neurologists study the brain and nervous systems, while linguists research how language structures can be instantiated in the brain. Psycholinguists are interested in language processing in normal individuals, while neuropsychologists research the breakdown of cognitive abilities due to brain damage. Speech-language pathologists provide therapy for language problems. Lastly, the type of neurolinguist of interest to this research is the cognitive scientist – a scholar involved in the studying of the processes involved in thinking and theories that may explain them. Cognitive scientists also research ways of using computer models to understand language performance (Obler and Gjerlow, 1999).

The two components of neurolinguistics can each be divided further into specific fields of study. Linguistics distinguishes between the following fields: phonology (the sound system), morphology (the system of meaningful units underlying words), syntax (the system for combining these units into sentences), discourse (the system combining sentences into larger meaningful utterances) and semantics (the system for understanding meaning). Neurologists, on the other hand, divide their study between the gross areas of the brain,

namely the cortex (external surface) and the subcortical area (larger internal space) (Obler and Gjerlow, 1999).

The field of neurolinguistics study is dominated by two major schools of thought, i.e. the localisationists and the connectionists. Localisationists believe that specific areas in the brain are responsible for language and study the effect that brain injuries on these areas have on language ability. The 'Boston School' is associated with the localisationists, following teachings by Norman Geschwind, Harold Goodglass and Edith Kaplan of the Aphasia Research Center in Boston. On the other hand, connectionists (or holists) argued that localisatinism falsely compartmentalises the language abilities of the brain that are, in fact, supported by larger parts of the brain. Holists argue that areas of the brain are interconnected. Dominent researchers in this camp include Hughlings Jackson and Kurt Goldstein (Obler and Gjerlow, 1999).

The holist approach is followed by those who design Parallel Distributed Processing models. These models are brain processing-based, comparable to computer networks. The holist approach appears to perform the same function that language does. According to this holist theory, there are no language centres, but rather network nodes, that are stimulated and, when stimulated enough, the node is realised, resulting perhaps as a spoken word. (Obler and Gjerlow, 1999).

2.1.2 Development of Language

Nature versus Nurture - are mental processes genetically bestowed or the result of experiences? Centuries of debate could not resolve this issue, not by Aristotle and Plato, nor by Locke and Leibnitz (Danks, 1981).

Language is often argued to be a genetic specification located in the human brain (Chomsky, 2002), and that humans are programmed specifically for language, to a level of detail that includes parts of speech distinction, how parts of speech relate to each other and even parts of grammar (McWhorter, 2004).

Chomsky has had several debates about human language development, most notably with Jean Piaget in October 1975 at the Abbaye de Royaumont near Paris (Piattelli-Palmarini, 1980). The main issue is nativism (Chomsky) versus constructivism (Piaget), thus the degree to which structure is genetically programmed. Although both agree that the biological and genetic foundations are substantial, they disagree as to the extent of the contribution. Chomsky argues that all concepts that are learned must pre-exist in the organism's structure (Chomsky, 2002; Danks, 1981). Contrary to this, Piaget holds that innate capacities exist as general mental strategies and that children use this to construct representations from their experiences. Danks (1981) puts forward that the debate was based on minimal empirical data, and that the reader would not likely be persuaded to change her view on the matter (Danks, 1981).

2.1.3 Grammar

Words are not just blurted out in isolation, but rather combined into phrases and sentences, where the meaning of the combination can be derived by the meaning of the words, and the way they are arranged (Pinker and Poeppel, 2000). Grammar is the set of rules inside everyone's head that specifies how words are arranged into meaningful combinations. It assembles words into phrases according to the words' part-of-speech categories, for instance, noun or verb.

Figure 2.1 shows the basic rules of grammar. As a first basic rule, an article (*a*) combined with a noun (*rose*) forms a noun phrase (NP). As a next rule, a verb phrase (VP) may consist of a verb (*is*) followed by its direct object, a noun phrase. The last rule reads that a sentence (S) may be composed of a noun

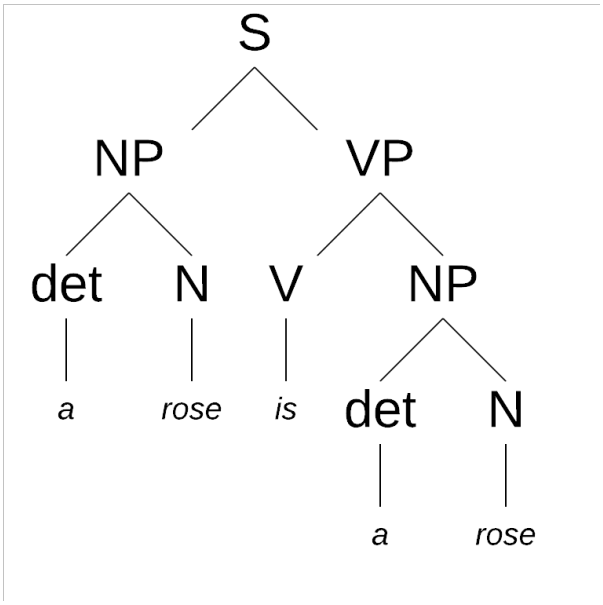

Figure 2.1: The basic rules of grammar shows how sentences (S), noun phrases (NP) and verb phrases (VP) are formed (Pinker and Poeppel, 2000).

phrase followed by a verb phrase. The figure displays building blocks of a complete toy grammar. Other rules specify meanings of new combinations and are beyond the scope of this research (Pinker and Poeppel, 2000).

2.1.4 Semantics

Semantics is the study of meaning as conveyed by elements or combinations of language. Utterances are not only scribbles or noises - they convey information (Swart, 1998). Pinker (2007) defines semantics as the relation of words to thoughts. Semantic research focuses on the relation between the linguistics expression used and what the expression refers to (object, properties, etc.), specifically on the meaning which arises out of the combination of more basic expressions into groups of words and sentences (Swart, 1998).

Consider the word *her* in the following examples:

(a) I took my daughter to the dentist this morning. She told her it wouldn't hurt too much, but she would give her a pain killer anyway.

(b) I took my daughter to the dentist this morning. She asked her if it would hurt and how long it would take.

(Swart, 1998).

Computational linguists thus have to combine linguistic with world knowledge to develop algorithms for anaphora resolution. Another important consideration for computational linguists is negation – the reversal of the truth value of the statement to which it is linked, for example, "It is raining" versus "It isn't raining".

Table 2.1: Natural Language Processing can be split into four categories, namely the application of it, the tasks, the methods, and the technology.

Application Why?	Task What?	Method How?	Technology Where?
• Sentiment Analysis • Chatbots • Natural Language Interface	• Part-of-Speech Tagging • Syntactic Parsing • Topic Modeling • Information Extraction • Text Generation and Translation	• Regular Expressions • Text Normalisation • Bag-of-Words • N-Grams • Term Frequency Inverse Document Frequency • Latent Dirichlet Allocation • Hidden Markov Models • Naïve Bayes • Deep Learning	• Packages • Frameworks • Services

2.2 Natural Language Processing

Dave Bowman: "Open the pod bay doors, HAL."
HAL: "I'm sorry Dave, I'm afraid I can't do that."
- Stanley Kubrick and Arthur C. Clarke, screenplay of 2001: A Space Odyssey

In the field of speech and language processing, it is important to firstly make a distinction between automatic speech recognition and natural language processing. Automatic speech recognition involves phonology and phonetics; the way words are pronounced in terms of the sequence of sounds, plus how the sound is created acoustically (Jurafsky and Martin, 2018).

Natural language processing, on the other hand, is defined as computational techniques used for analysing and representing natural text at several levels of linguistics analysis for the means of creating human-like language processing for a range of tasks (Liddy, 2001).

The topic of natural language processing (NLP) can be broken down into four categories, namely, the application of it (why NLP is done), the tasks (what is done in NLP), the methods (how NLP is done), and lastly the technology (where/in which frameworks NLP is done), as seen in Table 2.1. The logical flow of the topic is to firstly cover the applications, then focus and discuss some of the tasks used in the applications, moving to the methods used in the tasks and lastly the technology that exposes some of the methods. An example would be that the application sentiment analysis employs the task of topic modeling (to distinguish how the sentiment of one document is different from another); the method Term Frequency Inverse Document Frequency (TF-IDF) is used for topic modeling, and lastly the Natural Language Toolkit (NLTK) package exposes an easy implementation of the TF-IDF method.

Statistical Language Learning was one of the first books published on the topic of using statistical tools for natural language processing (Charniak, 1996). There were three main shortcomings with the book. Firstly, the book provided limited mathematical background. Secondly, the coverage of natural language processing

literature is criticised and, lastly, the book made substantial oversimplifications (Manning and Schütze, 1999). These shortcomings were addressed, with the publication of the definitive book "Foundation of Statistical Natural Language Processing", in which natural language processing is defined as all quantitative methods for automated language processing, for example, information theory, linear algebra and probabilistic modeling (Manning and Schütze, 1999).

2.2.1 Natural Language Processing Applications

Natural language processing is applied on a broad range of application. This literature review handles one of the core applications, which is sentiment analysis, plus two applications which are relevant to the topic of this research.

Sentiment Analysis

One of the most prominent forms of text categorisation is sentiment analysis. Sentiment analysis can be defined as sentiment extraction, positive or negative orientation of a writer towards an object (Jurafsky and Martin, 2018). Other researchers define it more broadly than just positive or negative, but as the computational study of human opinions, emotions, sentiments and attitudes towards entities such as organisations, services, products, individuals, events, topics or issues (Zhang et al., 2018). With the advent of social media, rating systems, blogs and reviews, the amount of text available to be analysed caused the field of sentiment analysis to grow rapidly within natural language processing. This is caused by the fact that opinions are central to human activities and behaviour. Sentiment analysis is used in fields such as marketing, finance and, lately, political science (Zhang et al., 2018). Sentiment analysis helps with the condensation of many different views on a topic or product. Some of the techniques for sentiment analysis include Naïve Bayes classifiers and neural networks, both discussed later.

Chatbots

Chatbots are programs that conduct a conversation through voice or textual methods (Shaikh et al., 2016). Chatbots are sometimes referred to as dialog systems, and one of the classic usages of chatbots is as a virtual assistant. Methods to achieve a human-like conversation include using dialog acts, and Part-of-Speech tagged tokens (Shaikh et al., 2016). In this research, deep learning will be explored as a method.

Frameworks such as LUIS.ai (discussed under Technology) use the concept of intents and entities. The chatbot will start with the user's utterance and from there firstly identify the user's intent, in other words, what does the user want to do? After the intent has been identified, the entities need to be extracted from the utterance. Consider the following two examples:

(a) User Utterance 1: "What time does the movie Braveheart start at Tygervalley?"

(b) User Utterance 2: "How much do two tickets cost for Braveheart at Tygervalley?"

The intent in user utterance 1 is to return a time; for this, the programmer would create code to handle this, for example, called: TimeIntent. In user utterance 2, the user wants to know the cost, and something like CostIntent would be able to handle that utterance.

After identification of intents, the entities need to be extracted from the utterance. For user utterance 1, it would be 'Braveheart' and 'Tygervalley'. Now the chatbot would have enough information to answer the

question. User utterance 2 shares these two entities, but also has an entity 'two tickets' which identifies the number of tickets (Berry, 2018). The intent and entity system is comparable to the dialog acts as intents and Part-of-Speech (POS) tagged tokens as entities (Shaikh et al., 2016).

Natural Language Interfaces

A natural language interface accepts query statements in natural language and sends data to a system which results in appropriate responses to the statements. The interface should be able to translate the natural language queries into actions (Chowdhury, 2003).

Up-to-date, simple, natural language queries have been the focus of natural language interfaces. However, recently, question-answering systems are being built that provide answers to natural language questions, rather than returning documents with information related to the question. Solutions focus beyond pure natural language processing, combining NLP with knowledge representation and information retrieval. This obtains domain independence and robustness regarding text variability (Chowdhury, 2003).

The biggest drawback of natural language interfaces comes from their weak interpretative power, caused by their inability to deal with human natural language nuances (Chowdhury, 2003).

2.2.2 Natural Language Processing Tasks

The topics handled in this literature review cover a range of tasks that are relevant to this research. Information extraction is perhaps the most important task as this is what needs to happen on a user's utterance in order to construct a query on a data warehouse. Several other tasks are discussed, as recommendations for future research depends on them.

Part-of-Speech Tagging

Dionysius Thrax of Alexandria (*c.* 100 B.C.) summarised the Greek of his day, describing syntax, diphthong, clitic and analogy. He also mentioned eight parts of speech which form the basis for most European languages: noun, verb, pronoun, preposition, adverb, conjunction, participle and article. Part-of-speech (POS) tagging is the task of assigning parts-of-speech to words (Jurafsky and Martin, 2018). POS tagging is defined as a dependency tree where the goal is predicting grammatical entities and relationships between them for each sentence or clause (Zeroual and Lakhouaja, 2018). This can be achieved with several methods, most notably Hidden Markov Models and Recurrent Neural Networks (RNN), both discussed later (Jurafsky and Martin, 2018).

The Penn Treebank Tagset (Marcus et al., 1993) has been used to label many corpuses. Consider the following example provided as training data in the treebank: The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS .

The part-of-speech is displayed after the slash in the treebank, using the abbreviations in Table 2.2.

Syntactic Parsing

Syntactic parsing consists of recognising a sentence and then assigning a syntactic formation to it (Jurafsky and Martin, 2018). The end goal of syntactic parsing is to create a tree structure of a sentence, while defining the POS tags as well. This involves the creation of algorithms that employ context-free grammars to produce the correct trees.

Table 2.2: Part-of-Speech tagging abbreviations, for example, *JJ* represents an adjective in a sentence (Jurafsky and Martin, 2018).

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	+,%, &	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

A classic example to understand the ambiguity involved in syntactic parsing is the following sentence: “I shot an elephant in my pajamas”. This sentence can be parsed in two ways, see Figure 2.2.

The first parsing had the elephant in the shooter's pajamas, whereas the second parsing has the shooter in his own pajamas. Disambiguation algorithms require semantic, statistical and contextual knowledge sources. An example of such an algorithm is the Cocke-Kasami Younger algorithm; lately, however, supervised machine learning approaches are being implemented (Jurafsky and Martin, 2018).

Topic Modeling

Collections of documents such as news articles or blog posts often need to be divided into natural groups in order to better understand or summarise them by a number of topics. Topic modeling is used for unsupervised classification of these documents. This is similar to clustering on numeric data, and it finds natural groups of items (Silge and Robinson, 2016). Latent Dirichlet Allocation is a popular method for topic modeling, discussed later (Blei et al., 2003).

Information Extraction

Information extraction turns the unstructured information in texts into structured data. The first step of information extraction is to find the named entities in text, called named entity recognition. This can consist of people, places, companies, protein names or financial asset classes. Deep learning algorithms are used for named entity recognition (Jurafsky and Martin, 2018).

Text Generation and Translation

Another natural language processing task worth mentioning is the automatic generation of text and translation, although the details of these topics are beyond the scope of this research. Text generation is the ability to generate realistic-sounding text from a knowledge base and is regularly modeled in two phases, namely,

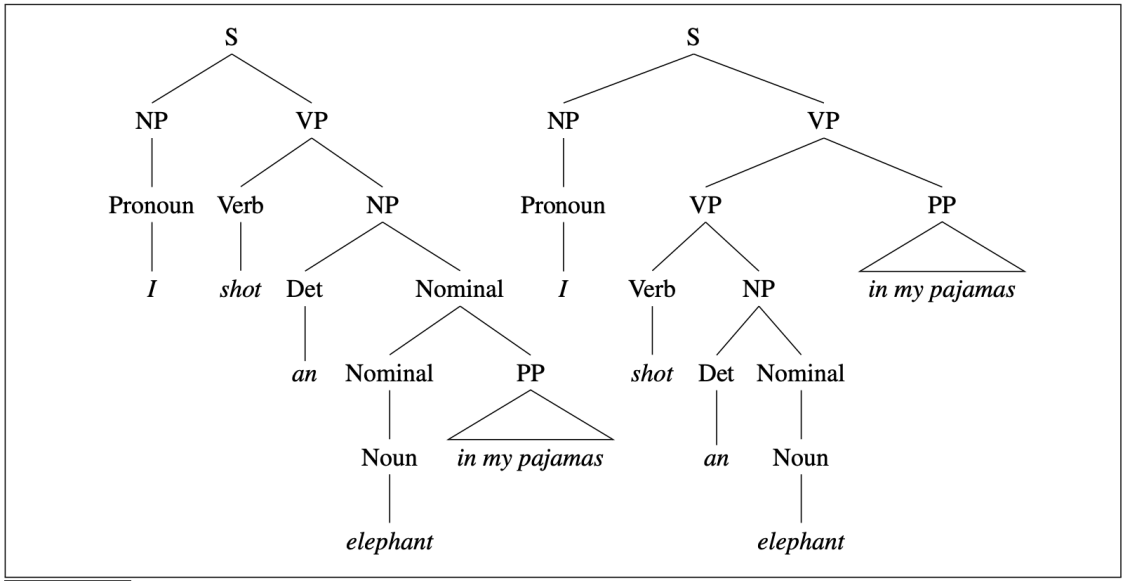

Figure 2.2: A parse tree showing how words in a sentence S "I shot an elephant in my pajamas" can be parsed into parts of speech (pronoun, verb, etc.) and then into phrases (NP = Noun phrase, VP = verb phrase). The sentence is ambiguous because it can be parsed into two different trees, each of which is syntactically correct (Jurafsky and Martin, 2018).

content planning (which is what to say), and sentence realisation (which is how to say it) (Jurafsky and Martin, 2018). Recently, Neural Machine Translation has replaced Statistical Machine Translation as the go-to method for automatic translation (Luong et al., 2015).

2.2.3 Natural Language Processing Methods

The selection of natural language processing methods was guided by the tasks which employ them. For instance, Latent Dirichlet Allocation is a popular method for the task of topic modeling. These methods do not cover all natural language processing-related methods; however, the first four methods (regular expressions, text normalisation, bag-of-words and n-grams) are all employed in the methodologies for this research. Four other topics were included to give a sense of important methods traditionally used for natural language processing. The order of the topics increases from very simple (regular expressions) to statistically rigorous (Naïve Bayes).

Regular Expressions

Regular expressions have for years been the work-horse of many text analysis projects, and are used in every word processor, computer language and text-processing tool. Regular expressions are algebraic notations used for the characterisation of a set of strings. Their strength lies in searching text, when a pattern has been defined to be used on a corpus of text (Jurafsky and Martin, 2018).

Text Normalisation

Normalisation of text is usually one of the first tasks of the natural language processing programmer. This means converting text to a more convenient, standard form. The first of these methods is tokenisation, which

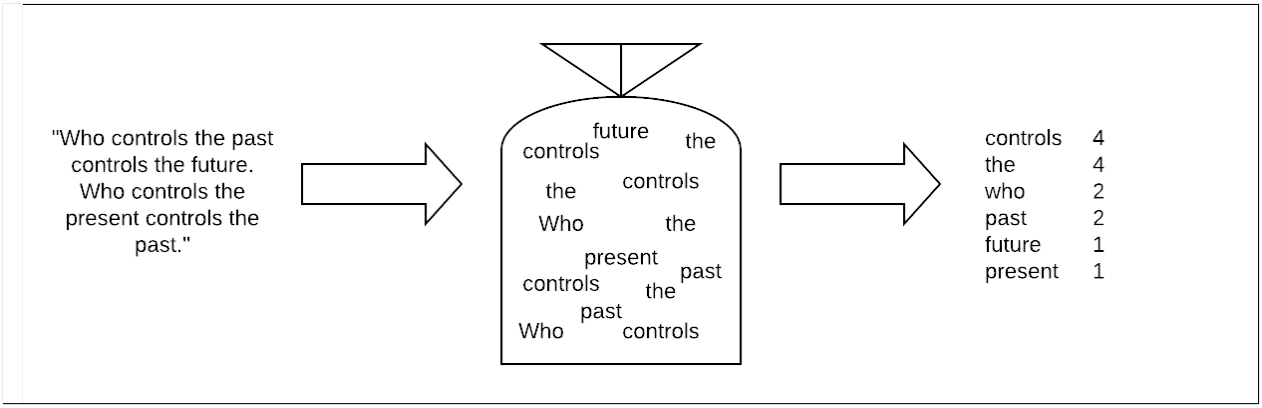

Figure 2.3: Illustration of a bag-of-words model. Ordered text (left-hand side) is broken into words and collected together without reference to order (middle), after which word frequencies are calculated (right-hand side).

means separating out words from running text. The next method, lemmatisation is the task of finding out if two different words have the same root. For instance, sang, sings, sung all have the same root, namely, sing. The last normalisation task discussed here is stemming. Stemming is a simpler form of lemmatisation where the suffix is simply stripped from the end of the word (Jurafsky and Martin, 2018).

Bag-of-words

One of the most basic representations of text is called the bag-of-words. A bag-of-words is an unordered set of words appearing, with no regard for position, in a document, keeping only the frequency of the number of times the word appears, as seen in Figure 2.3 (Jurafsky and Martin, 2018).

N-Grams

In language models, a sequence of N words is called an n-gram. A bigram is a two-word sequence, for example, 'I am', 'am very' or 'very lucky'. Along the same lines, a trigram consists of three consecutive words, for example, 'I am very' or 'am very lucky' (Jurafsky and Martin, 2018).

Term Frequency Inverse Document Frequency

Term frequency inverse document frequency is a simple method for finding differences in documents identifying themes of importance when comparing documents with each other. The term frequency inverse document frequency algorithm is the product of two terms:

- The term frequency: how many times the word appears in this document;
- The inverse document frequency: this is the number of documents in the corpus divided by the number of documents containing the word. The log function is normally used:

$$idf_i = log(\frac{N}{df_i}) \tag{2.1}$$

(Jurafsky and Martin, 2018).

Latent Dirichlet Allocation

Latent Dirichlet Allocation is used in the field of topic modeling by applying unsupervised learning to large sets of texts to induce sets of associated words. The similarity between Latent Dirichlet Allocation and the Naïve Bayes Classifier (discussed later) is that both assume a probabilistic model for documents (Jurafsky and Martin, 2018).

The following assumptions describe the premise for Latent Dirichlet Allocation:

- There is a fixed number of topics (k).
- There is a distribution (b_1) governing the probability of seeing word w given topic k.
- There is another distribution (b_2) governing the mixture of topics in document d.
- Each word in the document was generated by firstly randomly picking a topic from distribution b_2 and then randomly picking a word from distribution b_1.

(Blei et al., 2003; Grus, 2015).

Hidden Markov Models

For part-of-speech tagging, the Hidden Markov Model is one of the most used algorithms, as it is very useful for sequential data such as text. The actual words are generated from an underlying Markov chain of parts of speech. A Markov chain gives information on the probabilities of sequences of random variables, also known as states. Each state can take on values from some set, which can be words or symbols representing anything. One important feature of a Markov chain is the assumption that predictions on the future in the sequence only depend on the current state; no other states before the current one have an impact on future states.

$$Pr(Z_{t+1}|Z_t, ..., Z_1) = Pr(Z_{t+1}|Z_t). \tag{2.2}$$

(Zucchini et al., 2016).

A Hidden Markov Model is a probabilistic sequence model which computes a probability distribution over possible sequences of labels (the unobserved states, for example, parts of speech like nouns, verbs, etc.) given a sequence of units (the observed states, for example, words in a sentence), and then calculates the best label sequence (Jurafsky and Martin, 2018).

Consider the following tagged example: Janet/NNP will/MD back/VB the/DT bill/NN. In Figure 2.4, the discrete random variable Z_i is a finite set (m) of hidden variables. In the case of POS tagging, this represents the part-of-speech, for example, NNP or VB.

$$Z_1, ..., Z_n \in \{1, ..., m\} \tag{2.3}$$

The actual observed words are the random variable X_i; this could be real numbers or a discrete set, for instance, in the tagged example 'Janet' or 'back'. The following equation denotes real numbers, but in the field of natural language processing, it takes on a discrete set in the form of words.

$$X_1, ..., X_n \in \chi(\mathbb{R}) \tag{2.4}$$

A sentence or document is represented by d:

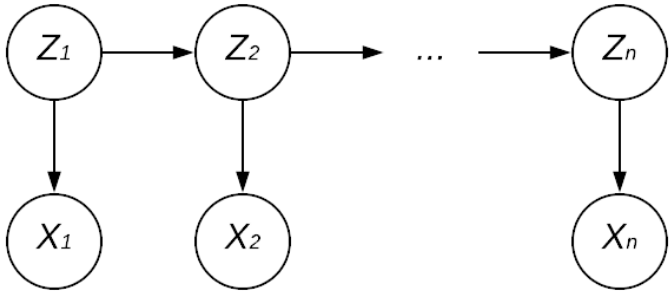

Figure 2.4: A Hidden Markov Model consists of hidden random variables Z_t and observed variables X_t.

$$d = (x_1, ..., x_n) \tag{2.5}$$

The joint distribution of random variables respects the trellis diagram in Figure 2.4. The joint distribution on all random variables factors in the following way: the probability of Z_1 times the probability of X_1 given Z_1, times the product of Z_k given Z_{k-1} times the probability of X_k given Z_k. In the first two words of the example, the probability of *NNP* being the starting tag times the probability of 'Janet' given *NNP* times the probability of *MD* given *NNP* times the probability of 'will' given *MB* and so forth.

$$P(x_1, ..., x_n, z_1, ..., z_n) = P(z_1)P(x_1|z_1) \prod_{k=2}^{n} P(z_k|z_{k-1})P(x_k|z_k) \tag{2.6}$$

Three variables are important when calculating the joint probabilities, namely the transition probabilities, the emission probabilities and the initial distribution.

Transition probabilities are a stochastic matrix that describes transitions between states, for example, words from *NNP* to *MD*.

$$T_{i,j} = P(Z_{k+1} = j|Z_k = i), i, j \in \{1, ..., m\} \tag{2.7}$$

Emission probabilities describe how the latent variables are emitted, for example, probability of the word 'Janet' occurring in the NNP state.

$$\varepsilon_i(x) = P(x|Z_k = i), i, j \in \{1, ..., m\}, x \in \chi \tag{2.8}$$

Initial distribution is the probability distribution for the first tag of a sentence, for example, the probability of the sentence starting with *NNP*.

$$\pi(i) = P(Z_1 = i), i \in \{1, ..., m\} \tag{2.9}$$

Using an algorithm like the Viterbi Algorithm, the initial distribution, the transition matrix and the emission matrix can be used to find the best part-of-speech tagging for a sentence (Ghahramani, 2001; Johnson and Willsky, 2013).

Naïve Bayes

Sentiment analysis is one of the most important types of text categorisation. One of the well-established traditional methods of text classification is Naïve Bayes. Naïve Bayes is a probabilistic classifier which, for a document d, out of classes $c \in C$, returns the class c which has the maximum posterior probability given the document (Jurafsky and Martin, 2018).

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) \tag{2.10}$$

Rooted in Bayes Rule, the conditional probabilities can be substituted into the original equation.

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{2.11}$$

This results in:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)} \tag{2.12}$$

The denominator $P(d)$ can be dropped, because the probability for each document is the same for each class; in other words, the most likely class is computed for each document. Thus, the class that maximises the equation is chosen.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c) \tag{2.13}$$

The document can be represented as a set of features $f_1, ..., f_n$.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(f_1, ..., f_n|c)P(c) \tag{2.14}$$

To make computation easier, Naïve Bayes adheres to the following simplifying assumptions: firstly, the bag-of-words assumption, in other words, the position of words does not matter. The second assumption is conditional independence; assuming that the probabilities of $P(f_i|c)$ are independent for class c and can be multiplied out. The Naïve Bayes classifier can thus be represented by:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c) \tag{2.15}$$

(Jurafsky and Martin, 2018).

2.2.4 Natural Language Processing Technology

When developing bespoke natural language processing solutions, several packages are available to the programmer. Cloud technologies enable several natural language processing frameworks and services. Frameworks will cover the offering by companies which can be used for general development or to build chatbots, whilst other natural language cloud services are also discussed.

Table 2.3: The four big cloud providers each offer a chatbot framework.

Vendor	Framework
Microsoft	LUIS.ai
Amazon	Lex
Facebook	Wit.ai
IBM	Watson

Packages

The Natural Language Toolkit is a Python platform for working with human language data, including libraries for stemming, tokenisation, tagging, parsing and semantic reasoning. The NLTK is community-driven, and a free, open-source resource (Bird and Loper, 2004). Other NLP packages for Python are offered by spaCy and Scikit-learn (Donaldson, 2018).

Frameworks

Table 2.3 covers a list of similar chatbot frameworks, each geared for rapid chatbot development and easy cloud deployment. The abilities and concepts of each of the offerings are more or less in line with each other. Each starts with a concept of the user utterance, moving to identifying the intent and then extracting the entities. The frameworks all implement machine learning and have easy exposure via Application Programming Interfaces (API).

SQuAD

Stanford University developed the Stanford Question Answering Dataset (SQuAD) to test question-answer natural language frameworks. SQuAD is a dataset of questions by crowdsourced workers on Wikipedia articles, where the answers are segments of text from the reading passage (Rajpurkar, 2018). Three of the best performing frameworks on SQuAD include BERT by Google (Devlin et al., 2018), nlnet by Microsoft and YARCS by IBM.

Services

The major cloud providers also offer APIs for natural language processing. Microsoft offers the Text Analytics API, with the following abilities: Sentiment analysis, Key phrase extraction, Language detection and Identify entities in your text (Microsoft, 2018). Amazon Web Services exposes the Amazon Comprehend services doing all of the following: Entities, Key Phrases, Language, Sentiment and Topics (Amazon, 2018). Lastly, Google has the Google Cloud Natural Language API with the following endpoints: analyzeEntities, analyzeEntitySentiment, analyzeSentiment, analyzeSyntax, annotateText and classifyText (Google, 2018).

2.3 Deep Learning

"I get very excited when we discover a way of making neural networks better - and when that's closely related to how the brain works." Geoffrey Hinton

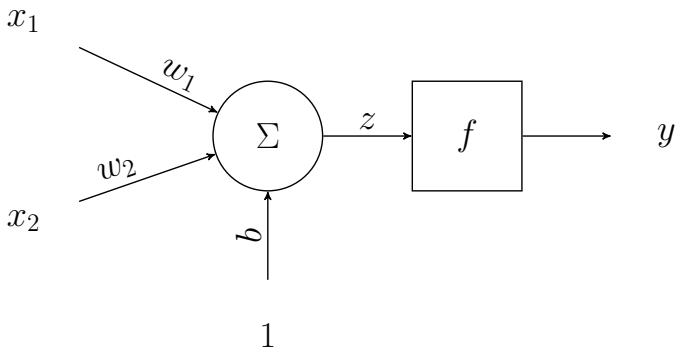

Figure 2.5: A single layer, feed forward neural network.

2.3.1 Neural Networks

The section starts off by discussing a single node, single layer, feed forward neural network. The single computational unit is the most basic building block of the neural network. A computational unit or neuron takes a set of numbers, performs some computation, and produces an output. As seen in Figure 2.5, the neuron is a weighted ($\boldsymbol{w}$) sum of inputs ($\boldsymbol{x}$) with an added bias term (b). This produces an output of a linear function of x, called Z.

$$z = b + \sum_i w_i x_i \tag{2.16}$$

A non-linear function f is applied to z, which is called the activation function. The sigmoid function depicted in Figure 2.6 and Equation 2.17, is a popular activation function. The Rectified Linear Unit function is another example of a popular activation function.

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.17}$$

Vector notation is mostly used, and the sum can then be replaced with the dot product.

$$y = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b) = \frac{1}{1 + exp(-(\boldsymbol{w} \cdot \boldsymbol{x} + b))} \tag{2.18}$$

(Jurafsky and Martin, 2018).

Layers

Deep learning consists of networks with many layers. In Figure 2.7, the input layer is marked with i_1 and i_2. There are two hidden layers, firstly h_1, h_2, h_3, h_4 and the second hidden layer with h_5, h_6, h_7, and h_8. Lastly, the output layer is noted by o_1 and o_2.

The training of neural networks consists of two steps, namely, the forward pass and then the backward pass. The forward pass happens when the input is taken thought the weights, nodes and activation function as described in the single node, single layer, feed forward example (Figure 2.5). After one set of observations has been taken through the network, one set of outputs are generated. These outputs can then be compared

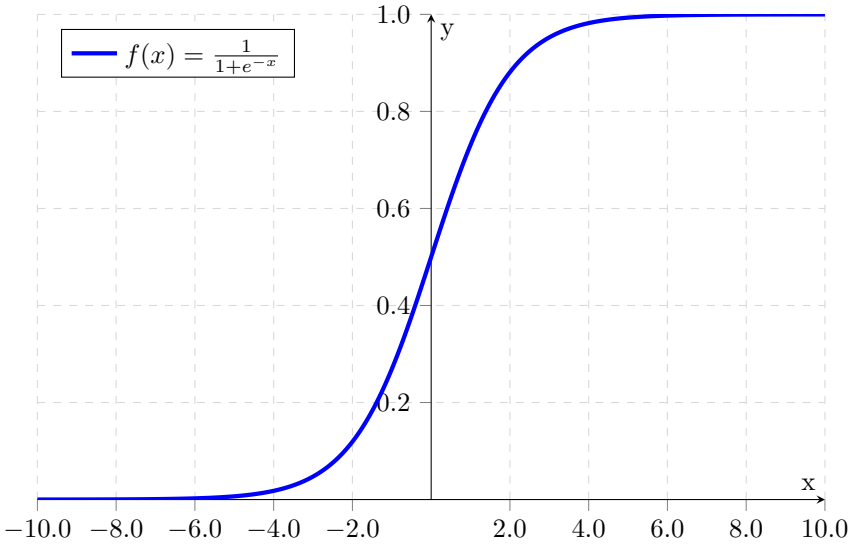

Figure 2.6: The Sigmoid function is a popular activation function.

with the target values by using a loss or error function. The backward pass (discussed later) updates the weights of the neural network.

A batch is the number of observations used to calculate one loss score before updating the weights in the backward pass. For example, if three observations are used to calculate a loss score, the batch size of the neural network is three.

The idea of epochs corresponds to the number of times all the observations are used for training. Two epochs would mean that all the observations are going to be used twice, independent of the batch size (LeCun et al., 2015; Mazur, 2015).

Stochastic Gradient Descent

Stochastic gradient descent is a popular optimiser for updating weights during the backward pass of the neural network. In this example, stochastic gradient descent will be discussed out of the context of neural networks, as it is just an optimiser that is used to calculate how much a loss function would change, if each parameter is changed by a small amount.

Consider the loss function depicted in Figure 2.8. Stochastic gradient descent tells us how to update the observed weight by taking the derivative of the loss function with respect to the parameter. In the graph it can be seen that if the slope of the derivative of the loss function at point w_1 is positive, the parameter will have to decrease to make the output of the loss function smaller.

The amount with which the parameter is decreased is governed by a learning rate. The learning rate makes the steps with which the parameter decreases smaller. The reason for this is that big jumps in the parameter could lead to the derivative changing a lot, for instance, with the next step having a negative slope (on the other side of the minimum).

The danger of being stuck in a local minimum rather than a global one always exists when using stochastic gradient descent (LeCun et al., 2015).

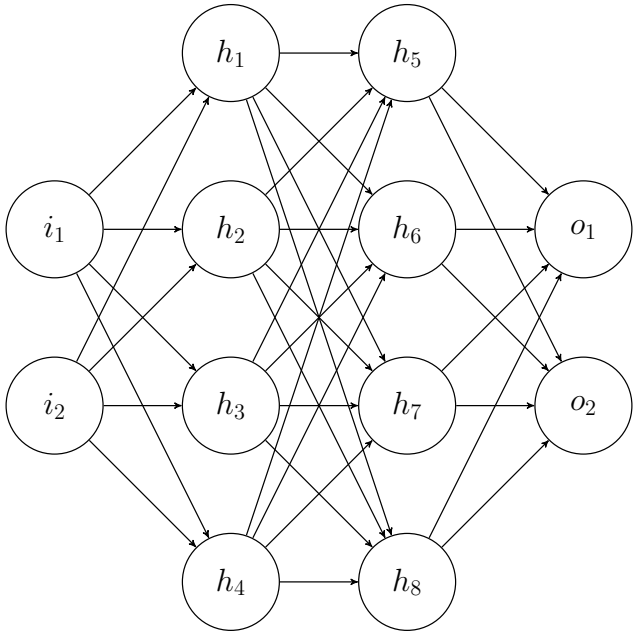

Figure 2.7: A multilayer neural network with an input layer with two nodes, two hidden layers with four nodes each and an output layer of two nodes.

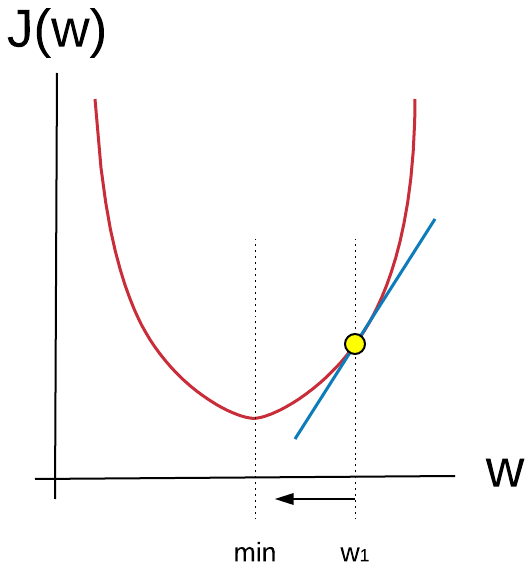

Figure 2.8: Stochastic gradient descent is an optimiser for updating weights during the backward pass of the neural network. The derivative of the loss function with respect to the parameter is taken, and indicates the direction in which the parameter needs to be adjusted.

Backpropagation

Backpropagation is the framework for the backward pass of the neural network training. This adjusts the weights of the network by using one of the optimisers, for example, stochastic gradient descent.

In the example of Figure 2.7, the all weights will have to be updated. When using stochastic gradient descent as the optimiser, the derivative of the loss function is taken with respect to each of the weights. Using the chain rule, it can be seen that w_{31} (the weight between h_5 and o_1) gets updated in the following way:

$$\frac{dE}{dw_{31}} = \frac{dE}{do_1} \cdot \frac{do_1}{dw_{31}} \tag{2.19}$$

This is useful, because by moving backwards through the graph, gradients can be calculated for any node based on the stored gradients from nodes that have already been visited (LeCun et al., 2015; Mazur, 2015).

2.3.2 Embedding

Vector representations of words are called word embeddings. One of the most prominent models for training embeddings is the word2vec model by Mikolov et al. (2013).

Machine learning methods for image recognition and speech recognition rely on data that are encoded in high-dimensional vector space, for example, the pixels of an image. All the data to train these models are encoded in these vectors. In contrast, natural language processing has traditionally handled words with arbitrarily assigned ids, for example, the word 'butter' with the id 123 and the word 'bread' with the id 456. The first disadvantage for handling text in this fashion is that it does not represent any relationship between the two words. It also leads to data sparsity, which requires a lot more data for training. Word embeddings seek to overcome these two disadvantages.

Vector space models map words in a high dimensional vector space, where semantically similar words, such as butter and margarine, are close to each other (Mikolov et al., 2013).

Word2vec

Word2vec learns word embeddings from raw text and is a very computationally-efficient predictive model. Word2vec allows two approaches: Continuous Bag-Of-Words or the Skip-Gram model. The models are very similar, with the main difference being that the Continuous Bag-Of-Words model predicts the target word (for example, 'mat') from the context (for example, 'the cat sits on the') and the Skip-Gram model predicts the context from the target word. Both models are trained using a binary classification objective (logistic regression) to discriminate the real target words v_t from noise words $\tilde{v}$, using the context word h. Only a subset (k) noise words ($\tilde{v}$) are selected from some noise distribution (P_{noise}), typically the unigram distribution. This has a computational advantage in that only k noise words are selected, and not all words in the vocabulary.

The objective is to maximise:

$$J_{\text{NEG}} = \log Q_\theta(D = 1|v_t, h) + k \mathop{\mathbb{E}}_{\tilde{v} \sim P_{\text{noise}}} [\log Q_\theta(D = 0|\tilde{v}, h)] \tag{2.20}$$

$Q_\theta(D = 1|v, h)$ is the binary logistic regression probability of seeing the word v in the context h in the dataset D, calculated in terms of the learned embedding vectors θ. This objective is maximised when high probabilities are assigned to real target words, and low probabilities to noise words.

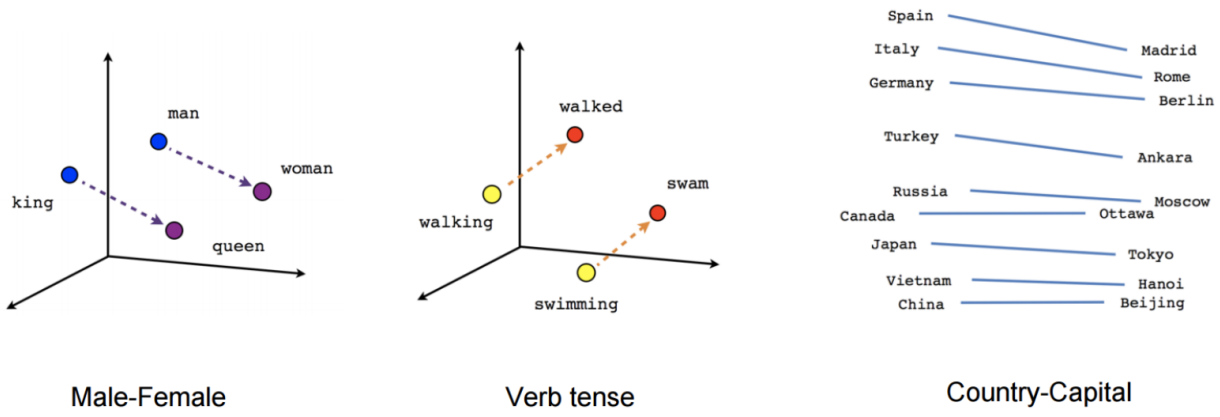

Figure 2.9: Once word embeddings have been trained, the relationship between words can be utilised mathematically. For example, similar distances can be seen between different countries and their capitals (Mikolov et al., 2013).

Consider the following example:
the quick brown fox jumped over the lazy dog

The first step is to create a dataset with one context word to the left and one context word to the right of the target word. This results in the following dataset:

([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), . . .

This is then converted to (input, output) pairs for the Skip-Gram model. This model tries to predict the context from the target word, and thus results in the following (input, output) pairs:

(quick, the), (quick, brown), (brown, quick), (brown, fox), (fox, brown), (fox, jumped), . . .

Using the first pair, the objective would be to predict 'the' from 'quick'. The noise word 'sheep' is drawn from a unigram distribution. Thus, the objective at time step t is:

$$J_{\text{NEG}}^{(t)} = \log Q_\theta(D = 1|\text{the, quick}) + \log(Q_\theta(D = 0|\text{sheep, quick})) \tag{2.21}$$

The goal is to update the embedding vectors θ so that the objective is maximised. This is done by deriving the gradient of the loss function with respect to parameters θ. The embeddings are then updated to take a small step in the direction of the gradient. This is then repeated over the entire dataset, until the model can distinguish between real target words and noise words.

The words are then represented by the multidimensional vectors. For visualisation, these vectors can be projected onto a two-dimensional space. It is then apparent that the relationships between the words are meaningful, for example, 'king' is to 'man' what 'queen' is to 'woman' in Figure 2.9 (Mikolov et al., 2013).

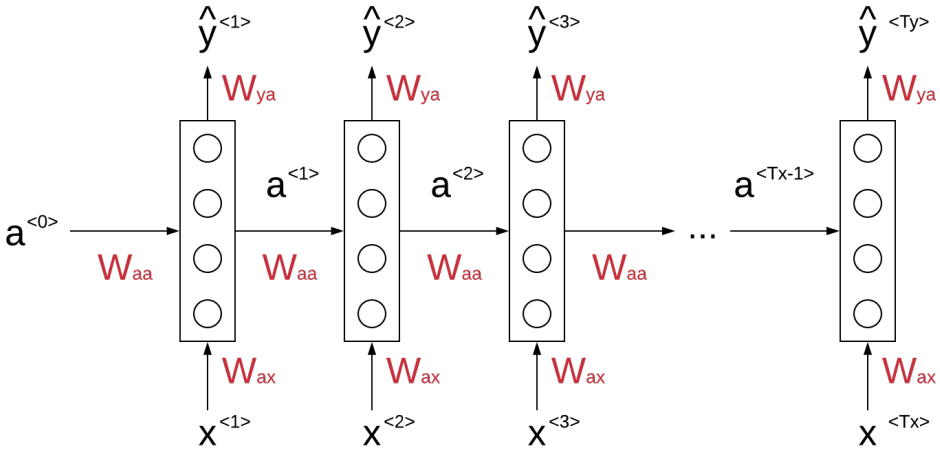

Figure 2.10: The weights (W_{aa}, W_{ax}, W_{ya}) in this Recurrent Neural Network architecture are the same across different timesteps.

Global Vectors for Word Representation

Another algorithm for training embeddings is Global Vectors for Word Representation (GloVe) by the Natural Language Processing Group at Stanford University (Pennington et al., 2014). GloVe is an unsupervised learning algorithm for getting word vector representations. The details of the algorithm are beyond the scope of this study; however, the group does expose pre-trained vectors. The global vectors were trained on several Wikipedia dumps. The result is that it has a vocabulary of words, with different size dimension vectors attached to it, which has already been trained, and useful (Pennington et al., 2014).

The main difference between GloVe and word2vec is that GloVe is a count-based model (based on word-word co-occurence counts), whereas word2vec is a predictive model which learns vectors to improve predictive ability of the target and context words (Pennington et al., 2014).

2.3.3 Recurrent Neural Networks

The reason why standard neural networks are not ideal for text analysis is that, firstly, text inputs and outputs can have different lengths. For example, with the task of information extraction (such as trying to identify people's names in a sentence), not all sentences have the same number of words. Secondly, the standard neural network does not share features learned across different positions in the text, for example, the name of the person does not always sit in word position number two.

Therefore, there are different types of neural network topologies that better fit sequence data like text, amongst others, Convolutional Neural Networks, Recurrent Neural Networks (RNNs) and Recursive Neural Networks (TreeNNs). Convolutional Neural Networks are beyond the scope of this research.

When dealing with sequence data, such as text, it is important to indicate the position or timestep of the input in the sequence; for this, the notation $<t>$ is used.

The task of information extraction is key to this research and was the base for the example to describe Recurrent Neural Networks. Consider the task of identifying country names from sentences, for example: "What are the sales for bikes in Germany?" From Figure 2.10, the $x^{<Tx>}$ are all the words in the sentence,

and the $y^{<Ty>}$ is the outcome, in this example: "$0, 0, 0, 0, 0, 0, 0, 1$".

There are several steps to the RNN, starting with feeding the first word (what) of the sentence to a neural network hidden layer, as seen in Figure 2.10. The neural network will then try to predict whether the word is a country name or not. The RNN then tries to predict the outcome of the second word (are) using the input $x^{<2>}$ and some information of what was computed at timestep <1>, that is the activation of timestep <1> which is represented by $a^{<1>}$ in Figure 2.10 at timestep <2>. This is then repeated until the last timestep. There is also an activation to start the process, depicted by $a^{<0>}$ in Figure 2.10. The same sets of parameters are used for each timestep. There are three sets of parameters, firstly W_{ax}, then W_{aa}, and lastly W_{ay}, as seen in Figure 2.10.

The forward propagation from left to right of the neural network would then be defined as:

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{2.22}$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y) \tag{2.23}$$

Next, a loss function is defined, and the backpropagation calculations are then carried backwards through the neural network, in effect doing backpropagation through time.

The architecture discussed in the example is for a Many-to-Many architecture, where $Tx = Ty$. In other words, the number of inputs is equal to the number of outputs. RNN architectures also exist for One-to-One, Many-to-One and One-to-Many. RNNs have several variants including long short-term memory units, gated recurrent units and Bidirectional RNNs, not discussed in this research (Ng, 2018).

2.3.4 Recursive Neural Networks

One important variant of the Recurrent Neural Network is the Recursive Neural Network (TreeNN). The underlying theory for using TreeNNs with language is that language consists of a recursive structure, and TreeNNs are meant for learning the structure of sequential data. Figure 2.12 shows that instead of doing the feed forward calculations of the neural network from left to right through time, a parse is used to combine sentence structures, and the calculations are done on that basis. This is to address one of the main weaknesses of RNNs, which is that the input towards the end of a sentence plays a bigger role than those in the front of the sentence.

Figure 2.11 shows that Recurrent Neural Networks often capture too much of the last word in the final vector, whereas Figure 2.12 shows that Recursive Neural Networks use the tree structure of the sentence (Socher, 2018). As can be seen in Figure 2.12, the backpropagation is done through structure instead of time (Socher, 2018). In other words, in a Recurrent Neural Network, backpropagation happens from the last word in a sequence backwards to the first word. In a Recursive Neural Network, backpropagation happens from the lowest structure ('the country') in the sequence towards the top structure.

2.4 Databases

"220 000 000 000 000 000 000 000"

According to the Cisco Global Cloud Index, in 2016, all people, machines and things created 220 zettabytes of data. The Index attributes this due to internet users doubling since 2011, the rise of social media, streaming

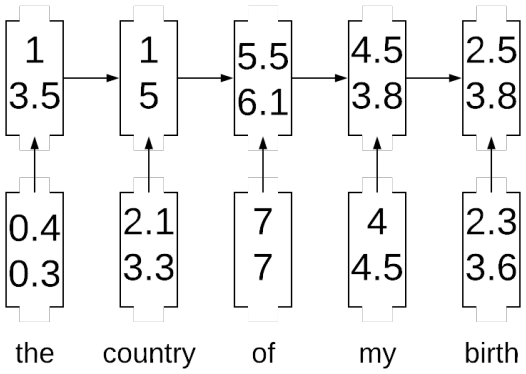

Figure 2.11: Architecture of a Recurrent Neural Network (RNN).

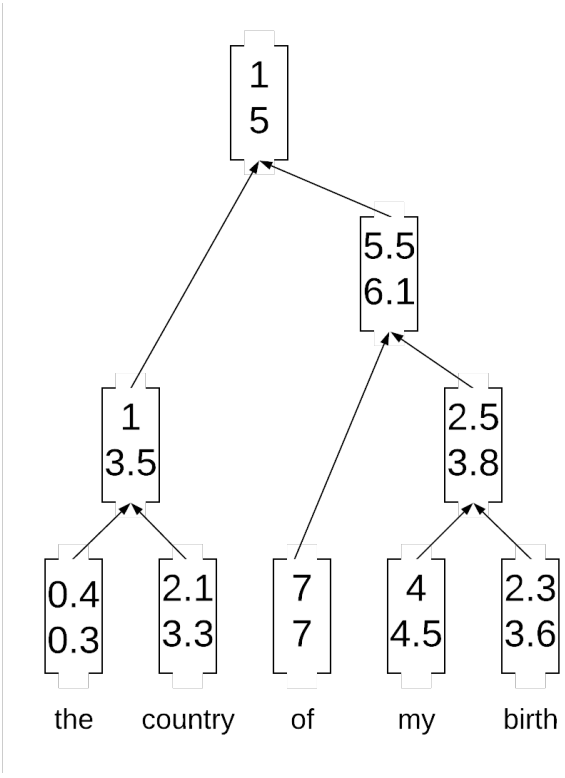

Figure 2.12: Architecture of a Recursive Neural Network (TreeNN).

service, the growth in mobile computing and the Internet of Things. The Index predicts that data creation will hit 850 zettabytes in the year 2021 (Lunt, 2018). Databases were introduced in the 1970s to address inflexible organisation of data and the fact that data were not easily accessible (Rob et al., 2006).

2.4.1 Relational Databases

The topics discussed for relational databases start off with a definition of what databases are. This research focuses on data warehouses, and the main goal of this section is to lead the reader to an understanding of what a data warehouse is. In order to achieve this, the reader must firstly understand what an operational database is. The underlying database theory, namely, the entity relational model, is then discussed. This section is followed by a discussion on normalisation theory, because normalisation is the main difference between an operational database and a data warehouse. Lastly, the mechanism to interact with both operational databases and data warehouses, namely, Structured Query Language, is discussed.

A database can be defined as an integrated, shared computer structure for storing end-user data or metadata (Rob et al., 2006). Relational databases can be classified into two main types, namely operational databases and data warehouses.

Operational Databases

Operational databases are also known as transactional databases and are used for a company's day-to-day operations. In contrast, data warehouses store data on information required to make tactical or strategic decisions (Rob et al., 2006).

Entity Relational Model

Peter Chen developed the entity relational model in 1975. It describes different kinds of relationships between entities with the help of entity relational diagrams. Relationships between entities can be one-to-one, one-to-many and many-to-many. In this model, an entity represents a real-world object, for example, a product. This is then represented in the database with a Product table. Each row in the Product table consists of details of a different product, for example, a mountain bike or a long sleeve jersey. The columns in the Product table can deal with different attributes of the Product, for example, the product name, product weight or product colour. Similarly, the real-world concept of a customer can be represented by a Customer table. Each row in the Customer table consists of a different customer, and the columns in the Customer table describes attributes of the customer, such as name or address. It is now easy to see that the Product can be related to the Customer table, because customers buy products (Rob et al., 2006).

Normalisation Theory

Normalisation theory extends from Entity Relational Modeling. Once entities have been modeled into tables, such as in the Product and Customer table example, normalisation takes care of finding optimal ways to design details of the tables. Normalisation consists of the evaluation and fixing of table structures in order to minimise data redundancies, thus reducing the chance of data anomalies (Rob et al., 2006). Normalisation goes through different stages, namely, first normal form, second normal form, third normal form, each building on the previous, with the third normal form being ideal for the majority of business database design purposes.

Table 2.4: A table which is not in third normal form, can have issues like different wording for the same item, see the spelling error in row two.

SalesId	Product	Sales
1	Classic Vest, S	200
2	Classic Vet, S	200
3	Classic Vest, S	300
4	Mountain-300 Black, 38	400
5	Mountain-300 Black, 38	400

Table 2.5: The Product table in third normal form only has one entry per item.

ProductId	Product
1	Classic Vest, S
2	Mountain-300 Black, 38

While the exact definition of each normal form is outside the scope of this study, it is necessary to handle an example to explain the general idea. The database table in Table 2.4 can be normalised by extracting the product out of the sales table to its own table. One of the advantages of normalisation can be seen in the example, as the second entry in the table is misspelled, which does not happen in the normalised schema (Rob et al., 2006).

The resulting schema would look like Table 2.5 and Table 2.6.

Data Warehouses

In the domain of data warehouses, there are two main influencers, namely Bill Inmon and Ralph Kimball. Inmon is seen as the "father of the data warehouse," and he defines a data warehouse as a subject-oriented, integrated, non-volatile, time-variant collection of data helping with decision-making (Rob et al., 2006).

The two authors advocate for different approaches to data warehouse development. Inmon promotes a top-down development approach which adapts traditional relational database tools for the development of a single enterprise data warehouse. In contrast, Kimball proposes a bottom-up approach using dimensional modeling, creating one data warehouse per major business process. Dimensional modeling is the task of splitting data into facts and dimensions (Breslin, 2004).

One of the disadvantages of normalisation is that it creates more relational join operations to produce output to the user, as can be seen in Table 2.5 and Table 2.6. Online analytical processing data warehouses are expected to perform fast analytics and are thus often denormalised back to first normal form or second normal form. This results in the creation of different schemas for data warehouses, namely the star-schema and snowflake-schema (Rob et al., 2006).

Table 2.6: The sales table in third normal form refers to the IDs of the products listed in the Product table, and not the wording of the products.

SalesId	ProductId	Sales
1	1	200
2	1	200
3	1	300
4	2	400
5	2	400

Table 2.7: SQL supports the create, read, update and delete operations.

CREATE	This is when an entry needs to be added to a table in the database.
READ	This entails selecting data from the database.
UPDATE	Rows in a database can be updated with this operation.
DELETE	This is used to delete rows in a database.

Two of the main concepts of data warehouses are that of facts and dimensions. Facts are values (numeric measures) which represent a particular business activity (Rob et al., 2006). Sales value or sales quantity is an example of a fact in a data warehouse. Dimensions are qualifying features for additional perspectives to a specific fact (Rob et al., 2006). The product that was sold, or the date on which it was sold, are examples of a dimension. It is important to note that a single dimension can be represented by a hierarchy, for example, the location where a product was sold can be represented by country, province, city and shop, all belonging to the location dimension. These facts and dimensions are represented in the data warehouse by physical tables. The fact table is related to each dimension in a one-to-many relationship.

In a star-schema, each dimension consists of only one table, opposed to the snowflake schema, where one dimension consists of multiple tables. In other words, the snowflake schema is a normalised form of the star schema.

Structured Query Language

The mechanism to access data in a database is through Structured Query Language (SQL) - a nonprocedural language (Rob et al., 2006). SQL provides create, read, update and delete abilities of data to the user. Table 2.7 details each operation.

2.4.2 Not Only SQL

With the advent of big data, demanding high reading and writing performance, traditional relational databases face new challenges. In large-scale and high concurrency applications (for example, search engines), the use of relational databases to store and query data has proven to be insufficient. Not Only SQL (NoSQL) was created to address this need (Han et al., 2011). One feature of NoSQL databases is that they are non-relational, meaning that the entity relational model which governs relational databases are not applicable here (Arora and Gupta, 2012).

NoSQL databases are motivated by the Consistency, Availability and Partition theorem (Han et al., 2011), which states that a distributed system can only meet two out of the three desirable properties of consistency, availability and tolerance of network partition. In this context, consistency means that the read operation will return the value of the last write operation. Availability means that every request to the database gets a response. Tolerance of network partition means the database can sustain any amount of network failure. The theorem states that a distributed system can only meet two out of the three at the same time (Han et al., 2011).

There are four different data models for NoSQL, namely key-value stores, document databases, column family databases and graph databases.

Key-value Stores

The data model of the key-value store simply means that a value links to a key. This simple structure, compared to relational databases, means that the query speed is higher, it supports mass storage and it supports high concurrency, meaning that lots of simultaneous connections can be made to it. A product called Redis is an example of a key-value store (Han et al., 2011).

Document Databases

Similar to a key-value store, the document database differs only in that the value of the document database is semantic and is stored in JavaScript Object Notation or Extensible Markup Language. An example of a document database is MongoDB (Han et al., 2011).

Column Family Databases

The table is used as the data model for column-oriented databases; however, the table association is not supported. The defining characteristic is that data are stored separately for each column. Cassandra is an open source column-oriented database, used by Facebook (Han et al., 2011).

Graph Databases

Graph databases expose the ability to traverse nodes and their relationships. Neo4j is an example of a graph database (Robinson et al., 2013).

2.4.3 Cloud-based Databases

The progression beyond NoSQL was for Cloud providers to offer Database as a Service. This resulted in Data as a Service, which allows users to store data at a remote disk in the cloud. Database as a Service is one step beyond Data as a Service, in that it offers database functionality and allows for the storage and access of databases at remote disks through the internet. Examples are Amazon RDS, Google's BigTable and Microsoft's SQL Azure (Arora and Gupta, 2012). Cloud offerings also include NoSQL options, for example, Amazon's DynamoDB, which is the Amazon Web Service version of MongoDB or Amazon Neptune, a cloud-based graph database (Amazon, 2018).

2.5 Natural Language Processing on Databases

This research concerns natural language processing methods on data warehouses, a subset of databases. This section is divided into four topics, starting with natural language interfaces on databases which cover non-neural network-related topics on databases. This moves to the neural network-related research on databases. Next, the scope of the literature is narrowed to embedding being applied on databases, and lastly, the use of the word2vec algorithm on databases is discussed to coincide with the methods employed in this research as close as possible.

2.5.1 Natural Language Interfaces to Databases

Natural language interfaces to databases have the goal of translating a natural language query to some query language, mostly SQL. Such a system combines three elements: firstly, the query interpretation part, then

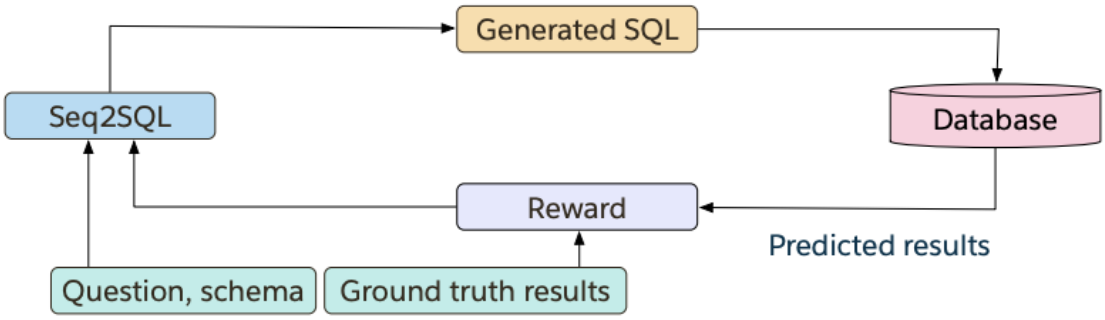

Figure 2.13: The Seq2SQL algorithm uses reinforcement learning to generate structured queries (Zhong et al., 2017).

the interactive communicator, and, lastly the query tree translator. The query interpretation part includes a parse tree node mapper and a structure adjustor; this represents the natural language query in a query tree. The next part, known as the interactive communicator, interacts with the user to guarantee that the interpretation is correct. Lastly, the query tree translator generates a SQL statement to be executed against a database. The heart of this system is the query interpretation part, which identifies nodes in the linguistics parse tree that can be mapped to SQL components, which will then be tokenised (Li and Jagadish, 2016).

Another system does not include the interaction with the user, but starts with some pre-processing to take care of simple text tasks such as white space removal and taking care of negation. In this system information extraction is used to locate named entities from the text into predefined categories, for example, names of people. Once entities have been recognised, the query can easily be generated (Falle et al., 2017).

2.5.2 Deep Learning on Databases

Salesforce developed and released the Seq2SQL algorithm as open source in 2017. Seq2SQL is a framework for generating structured queries from natural language using reinforcement learning. The model is a deep neural network for taking natural language questions and turning them into corresponding SQL queries. It uses rewards from in-the-loop execution of database queries and learns a policy to create the SQL query. The input into the model is the question and the columns of the table. A SQL query is then created and executed. The reinforcement learning algorithm is then trained using the result of the query execution as the reward, as seen in Figure 2.13 (Zhong et al., 2017). A main difference between the Seq2SQL model and this research is that the Seq2SQL model does not train vectors of words in the database, but rather maps a question to a query.

2.5.3 Embedding on Databases

In 1999, SAP registered a patent for a search engine that ranks documents in a database using rules that match characteristics of the database with a natural language query. For this patent, no word2vec training is involved, which is a big difference to this research. The vectors for the patent are updated according to an evolutionary algorithm using user feedback (Kaiser, 1999). Another architecture, called Sequence-to-sequence has been used to model the conditional probability of a SQL query, given the natural language utterance.

The supervised system was developed on 170 000 pairs of SQL/questions. The encoder and decoder were long short-term memory cells with two hidden layers and 500 neurons, plus the word embedding layer also had 500 neurons. The main difference between the sequence-to-sequence model and this research is that the sequence-to-sequence model will try to predict a SQL query given some text, whereas this research focused on building a query after identifying database terms in the utterance (Brad et al., 2017).

2.5.4 Word2vec on Databases

An example of where word2vec has been applied on databases is in a system called the Cognitive Database (Bordawekar et al., 2017). Meaningful vectors are created through unsupervised learning using database text. Syntactic and semantic characteristics of database tokens are captured in the vectors, which are then used to enhance database queries. A new class of queries called cognitive intelligence is then possible. As an example, in a sales database, similar buying patterns can be queried by looking for similar relationships between frequently bought products. In other words, find a vector that is the difference in vectors between peanut-butter and jam, then find all other product combinations that have the same magnitude and direction as the peanut-butter and jam vector (Bordawekar et al., 2017). The difference to this research is that this is not a natural language interface which will generate SQL from a user's utterance, but rather a system to enhance new query capabilities on a database. In other words, the user still has to create the query in SQL, but now has new user-defined functions to call and enable cognitive queries.

One application of using word2vec on databases is in the medical field. Embedding was created on a medical relational database in order to solve the problem medical researcher face when having to search through scientific papers and databases (Hyland et al., 2016).

WordNet is a lexical database of English developed by Princeton University (Lee et al., 2016). Several studies have applied word2vec and GloVe on this database. It was done for measuring semantic relatedness (Lee et al., 2016) and to derive better initialisation for training relational models (Long et al., 2016). Both these techniques created tools that is used for further natural language processing; in other words, the embedding was not done on a database outside of natural language processing, such as a sales database, and that is the difference to this research.

Chapter 3

Methodology

3.1 Adventureworks Data

The Adventureworks dataset is an open source dataset, licensed to be used for academic and commercial projects without licence. The data is on a fictional bicycle company, which sells multiple product categories across the globe.

The data for this research has been formatted into a star schema data warehouse. It has been denormalised into one flat file dataset. The sales fact in the middle of the schema consists of the sales value and the sales quantity. The three classic dimensions are date, location and item. The location dimension consists of a hierarchy which goes from shop at the lowest grain through to city, province, to country at the highest level. Similarly, the item dimension is also a hierarchy with product at the lowest level, followed by sub-category and then having category as the highest level. An extract of the data can be seen in Table 1.2.

3.2 Natural Language Processing on Data Warehouse System

A system capable of handling natural language processing on data warehouses consists of two inputs and one output (see Figure 3.1). The first input is the data warehouse on which natural language queries is to be performed. The second input is the query asked by the user. The system then computes the answer and the output of the system is the answer provided by the system to the user.

3.3 Method: Heuristics Model

The main purpose of the heuristics model is to identify which fact the user wants from the data warehouse, and by which dimensions the result needs to be filtered. To find the fact is the easier task of the two as, in most cases, the user will mention the column of the fact in their utterance, for example: in the utterance "What is the sales value for mountain bike socks in Georgia for last week?" The three principal dimensions in the data warehouse are the date, the location and the item. To filter date, simple regular expressions can be used, so the biggest difficulty for the model is to get the location and item. Both these consists of a hierarchy of several columns each: location with country, province, city and shop, and item with category, sub-category and product.

Thus, to extract how the query needs to be filtered for location, it needs not only to find the value of the location, but also the type of the location. In case of the "What is the sales value for mountain bike socks

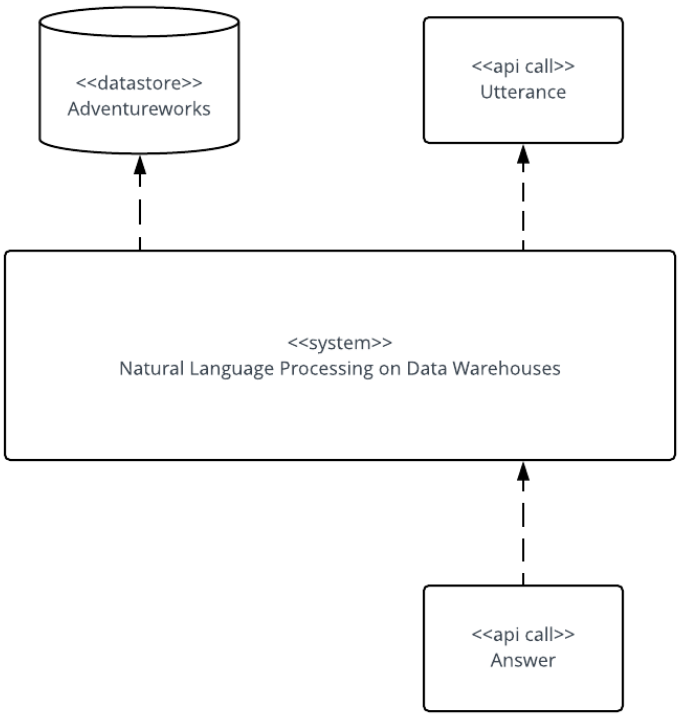

Figure 3.1: A natural language processing system on data warehouses takes a data warehouse and a user utterance as input, and send the answer as an output.

in Georgia for last week?" utterance, the value for location is Georgia, and the type of location is province. Per hierarchy, the model follows the following heuristic algorithm:

Get the longest number of consecutive words from the utterance at the highest level of the hierarchy.

The same algorithm is followed for the item dimension. In the example, it might find a match for 'mountain bike' at the sub-category level, but 'mountain bike socks' is a longer string, thus beating 'mountain bike'. However, 'mountain bike socks' appears only at the product level; if something similarly named appeared higher up in the hierarchy, that value and column would have been selected. One caveat of the heuristics model is that the spelling and word order needs to be exactly as in the data warehouse for it to be matched.

3.3.1 Heuristics System

The whole system is divided into two separate sub-systems, depicted in Figure 3.2. Firstly, data artifacts need to be created to store unique values per column; so, the *A1. Preparation* sub-system consists of only one component, being *A1.1 Create Unique Values for Columns*.

The *A2. Querying* sub-system consists of two components, firstly, *A2.1 Construct Query with Heuristics*, which uses the 'longest sentence at the highest hierarchy level' algorithm, and, lastly, *A2.2 Execute Query* to send the query to the data warehouse and return the result to the user.

3.3.2 Sub-system A1. Preparation

Sub-system *A1. Preparation* runs before and independently of the user query flow. Once a data warehouse has been identified, this sub-system is only responsible for creating a unique value for each hierarchy column.

Component A1.1 Create Unique Values for Columns

Component *A1.1 Create Unique Values for Columns* is a simple procedure to select unique values for each column in the location and the item hierarchy. The result is stored in a simple data artifact to be used later when the user utterance is sent to the system.

3.3.3 Sub-system A2. Querying

The *A2. Querying* sub-system runs only when the user sends an utterance to the system. It consists of two components, firstly to create the SQL, and, secondly, to run and return the query.

Component A2.1 Construct Query with Heuristics

For the SQL query to be constructed, component *A2.1 Construct Query with Heuristics* needs to return the chosen fact and the dimensions. The fact is what the query needs to return from the data warehouse, and the dimensions are used to filter the query. The fact usually consists of a selection of either the sales value or the sales quantity, and the dimensions consist of any combination of the date, item and location. The order in which the facts and dimension are extracted from the user utterance is as follows: sales fact, date, location, item. The sales fact and date dimension are extracted with heuristics, and the location and item use the heuristics algorithm and artifacts.

Firstly, to extract the sales fact, a simple algorithm will check for words indicating whether the user wants to see the sales value or the sales quantity. Variations (to cater for plural, etcetera) and synonyms of 'value' and 'quantity' are used to extract the fact information, after which the sales value part is removed from the

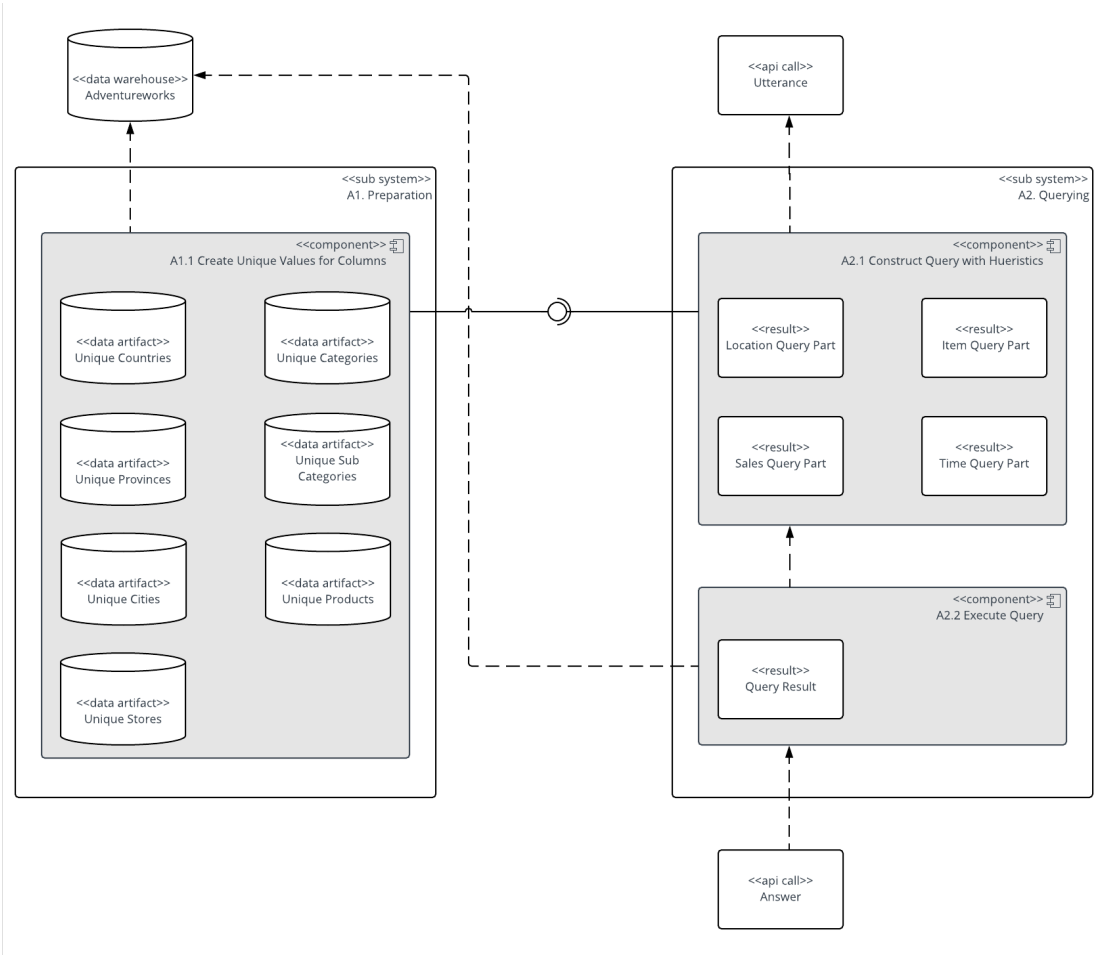

Figure 3.2: The heuristics model consists of two sub-systems, namely, the preparation and querying sub-systems.

utterance. For instance, starting with "What is the sales value for mountain bike socks at Georgia for last week?", would continue as "What is the for mountain bike socks at Georgia for last week?" Provision is also made for when the user wants a list of any of the columns in the data warehouse, for example: "What are the top countries for mountain bike socks?" This will list the top selling countries in terms of mountain bike sock sales, and the regular expression looks for the keyword 'top' in this case.

Next, the dates are extracted from the user utterance, once again by using heuristics. Different date formats are catered for by using regular expressions, for instance, looking at 2018/01/01 or 31 December 2018. Date-related concepts like 'yesterday' or 'last week' are also catered for with simple text filtering. This is then converted to date formats. This part of the component caters for one or two dates, as the user can ask for sales between two specific dates. The date parts are then removed from the user utterance before the next step. When no dates are found, a default of yesterday is populated. This leaves the next step with "What is the for mountain bike socks at Georgia for" from the example.

The next step is getting the location part of the query, and this is where the output of component *A1.1 Create Unique Values for Columns* is used. The 'longest consecutive number of words at the highest hierarchy level' algorithm is used here. To start the algorithm, the word 'what' is looked for in the unique country values data artifact, followed by searching for it in the unique province data artifact, then the unique city data artifact, and lastly in the unique shop data artifact. If the algorithm finds a match at any point, it stops, and does not move to the next level in the hierarchy. (For example, the word 'united' will be found in the unique country data artifact, so 'united' will never be tested in province, city or shop.) Back to using the "What is the for mountain bike socks at Georgia for" example, the words 'What is' will be tested next in the algorithm from country through to shop. It will continue to run all consecutive combinations of the words until it is done. The algorithm will only replace the selected value and column of the length of the string if the latest match is more than the existing selected value. Once the value and column for location has been found, it is removed from the utterance, thus leaving "What is the for mountain bike socks at for".

Lastly, the same algorithm is applied to look for the item column and value. In the mentioned example, again, the word 'What' is looked for in the item hierarchy, starting with the category, then the sub_category, and lastly the product columns. This is followed by 'What is', and so forth. The algorithm will eventually find 'mountain bike' in the category level, however, 'mountain bike socks' will be found at the product level, and since the string for 'mountain bike socks' is longer than 'mountain bike', this will be the chosen match, even though category is at a higher level than product.

Once the fact and three dimensions have been established, the query is constructed into one SQL command.

Component A2.2 Execute Query

The SQL command output from component *A2.1 Construct Query with Heuristics* is then sent to the data warehouse, and the result returned to the user in component *A2.2 Execute Query*.

3.3.4 Problems with the Heuristics Model

There are two major problems with the heuristics model. The first problem is that the order of words has to be the same in the utterance of the user and the column in the data warehouse; for example, if the user asks for 'green long sleeve jersey' and the data warehouse describes the product as 'long sleeve jersey green', there

would be no match. A possible way to address this problem is to test every single combination of the four words. However, this leads to the second issue of the model, and that is that the heuristics algorithm takes long to compute. The example would contain more than just the item words, in that stop words are still included, in other words "What is the of long sleeve jersey green" could take up to $8! \times 200000$ iterations in a modern data warehouse containing 200 000 items. The embedding model addresses both these problems.

3.4 Method: Embedding Model

Before addressing the details of the embedding model, it is worth looking at the structure of the data warehouse. It consists of facts and dimensions. The two fact columns are the value and the quantity for sales. The first dimension is the date dimension, represented by one column. The other two dimensions are the location dimension and the item dimension, each represented by several columns, organised in a hierarchy. The location dimension is represent by four columns in the hierarchy, starting with the top level of country, moving down to province, then down to city, and lastly to shop. Likewise, the item dimension consists of three columns arranged in a hierarchy, starting with the category at the top level, moving down to the sub-category level, and lastly to the product level. The main aim for the embedding model is to organise the text in the location and item hierarchies in such a way so that it is easy to spot the values in an utterance. To that effect, for the embedding model, three different types of vectors are created: **word vectors**, **column vectors** and **utterance vectors**. The word vectors are trained using word2vec, initiated by global vectors. An important key insight is that there is one set of vectors trained for each word in the location hierarchy, and a different set of vectors for the item hierarchy. Some words overlap in both hierarchies for example, the word 'bike', and one will then have a vector trained from the location hierarchy, and one trained from the item hierarchy. The column vectors are then created for each unique entry in each column of the location hierarchy by calculating the average of the location-based word vectors. The same happens for the columns in the item hierarchy: each unique value in each column in the item hierarchy gets calculated by taking the average of the item-based word vectors. The utterance vectors are calculated by firstly taking the location-based word vectors and creating one average vector for the whole utterance. The nearest unique location-based vector is then determined by using the cosine similarity:

$$cos(\theta) = cos(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{||\boldsymbol{a}||||\boldsymbol{b}||} = \frac{\sum_1^n (a_i \times b_i)}{\sqrt{\sum_1^n (a_i^2)} \times \sqrt{\sum_1^n (b_i^2)}} \tag{3.1}$$

(Hu et al., 2018).

Similarly, the utterance vector from the perspective of the item-related word vectors are created by taking the average of the utterance words by using item-related word vectors. Again, by using the cosine similarity, the nearest item-based vector is matched to the item based utterance vector. Consider the following example.

Word Vector of 'bike' from the location hierarchy:

$[-0.0929, -0.0027, 0.0763, ..., -0.0452, -0.0024, 0.1719]$

Word Vector of ‘bike’ from item hierarchy:

$[0.0870, -0.0425, -0.0467, ..., -0.0531, 0.2647, 0.0593]$

Next, a vector representation of each unique entry (a combination of words) in each column in the location and item hierarchies is calculated by averaging over the word vectors of each word making up the entry. For example, a vector representation for ‘mountain bike socks’ is computed by averaging the word vectors for ‘mountain’, ‘bike’ and ‘socks’. Some words like ‘bike’ appear in both the location and item dimensions. A word vector exists for each word in each dimension or context, thus requiring some way of knowing which word vector to use for ‘bike’. This is easily resolved; the column is known in which an entry appears, thus showing which hierarchy is being referred to. For example, it is known that because ‘mountain bike socks’ appears as an entry in the product column, it is associated with the item hierarchy, and therefore word vectors for ‘mountain’, ‘bike’ and ‘socks’ should all be sourced from the item-based word vectors. To further the example:

Average of ‘mountain’, ‘bike’ and ‘socks’ using the item hierarchy trained word vectors:

$[-0.0651, -0.0733, 0.1713, ..., -0.0218, 0.0945, 0.0587]$

Average of ‘better’, ‘bike’ and ‘shop’ using the location hierarchy trained word vectors:

$[0.0066, 0.0629, 0.0754, ..., -0.0636, 0.1675, -0.0562]$

Next, two average vectors for the user utterance gets calculated, one using the location word vectors, and one using the item word vector. The item-based average vector for the utterance is computed from the average of the item-based word vectors for ‘mountain’, ‘bike’ and ‘socks’, as other words in the utterance either do not appear in the item hierarchy (‘Georgia’, ‘sales’) or are stop words (‘what’, ‘are’, etcetera). For the utterance: “What are the sales for mountain bike socks in Georgia?” the following is computed:

Average vector for ‘mountain’, ‘bike’ and ‘socks’ according to the item hierarchy (there is no vector for ‘Georgia’ in the item hierarchy):

$[-0.0651, -0.0733, 0.1713, ..., -0.0218, 0.0945, 0.0587]$

Average vector for ‘bike’ and ‘Georgia’ using the location hierarchy (there are no vectors for ‘mountain’ and ‘socks’ in the location hierarchy):

$[-0.1105, 0.1013, -0.0600, ..., 0.0839, 0.1772, -0.0167]$

For each column in the item hierarchy (product, category, sub-category), the cosine similarity is computed between the item-based utterance vector and each of the entries in that column. Those entries can then be identified (regardless of column) that are most similar to the utterance. For example, it might be found that the column vector that is most similar to the utterance vector is ‘mountain bike sock pair’ in the product

category, followed by 'mountain bike footwear' in the sub-category level. This similarity tells the system two things: firstly, that the column to be used is the product column, and secondly that the value to look up in that column is 'mountain bike socks'.

In the case of the location example, the cosine similarity between the average vector for 'bike' and 'Georgia' brings back the closest match of the province 'Georgia'. The use of 'bike' from the item did skew the cosine similarity to not be a perfect fit with 'Georgia', but the vector for 'Georgia' was still influential enough to result in the highest cosine similarity to the utterance. The result is that the chosen column is province, and the value to look up in that column is 'Georgia'.

3.4.1 Embedding System

In terms of putting the described methodology together, the components for the embedding model can be split into two major sub-systems. The first sub-system entails the training of the models and the second sub-system is concerned with using the trained models to convert an utterance to a query.

The training sub-system consists of three components. The first component is to create training data. This component is dependent on a datastore to provide training data. This is followed by the embedding component, which trains the word vectors. The last component in the training sub-system is the creation of the average column vectors for each unique entry in each column.

The querying sub-system consists of four components. The first component uses the unique word vectors of the training sub-system and converts the utterance by the user, creating the utterance vectors. The next component takes the converted utterance vectors and gets the nearest column vectors by using cosine similarity. Thirdly, the final query is created and, finally, the last component executes the query on the data warehouse and sends the result to the user, see Figure 3.3.

3.4.2 Sub-system B1. Training

The first sub-system functions to generate data artifacts with which the user utterance can be evaluated. This involves generating training data, training the vectors for words in the data warehouse, and constructing average vectors for the column in the data warehouse.

Component B1.1 Create Training Data

The first component in the system is responsible for reading data from the data warehouse (input of the component) and creating training data (output of the component) for the word2vec algorithm (Vasudevan and Wicke, 2018). This is illustrated in Figure 3.4.

The word2vec algorithm relies on co-location of words (normally derived from sentences in text), so in order to train data from the cell entries in the data warehouse, all the columns per row are concatenated into a 'sentence'. Inspecting one row, the column for country appears next to the column for province, and so forth. As an example, to create the training 'sentence', 'united states' would be combined with 'new york' to form a training 'sentence' consisting of 'united states new york', followed by the other columns in that single row. After the row has been concatenated into one 'sentence', the rows are then concatenated into one big corpus. A key insight into the success of the system is the way in which training data is grouped, in other words, which columns are grouped to form 'sentences'. Different configurations have been tested, ranging from generating one set of training data to be fed to the word2vec algorithm, to a set of training data per

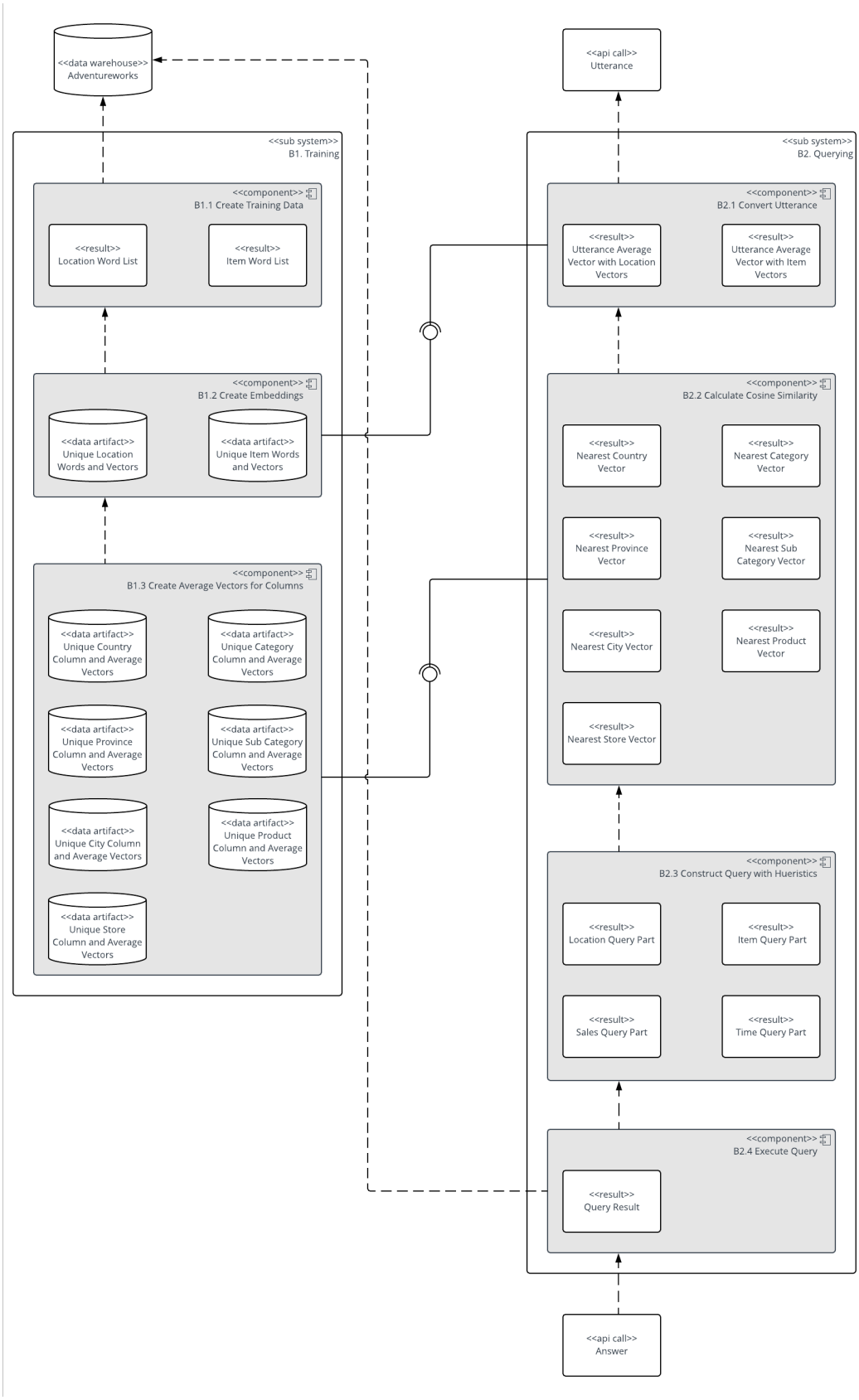

Figure 3.3: An embedding system consists of two sub-systems, namely the training and the querying sub-systems. The main goal of the training sub-system is to create data artifacts that is used by the querying sub-system.

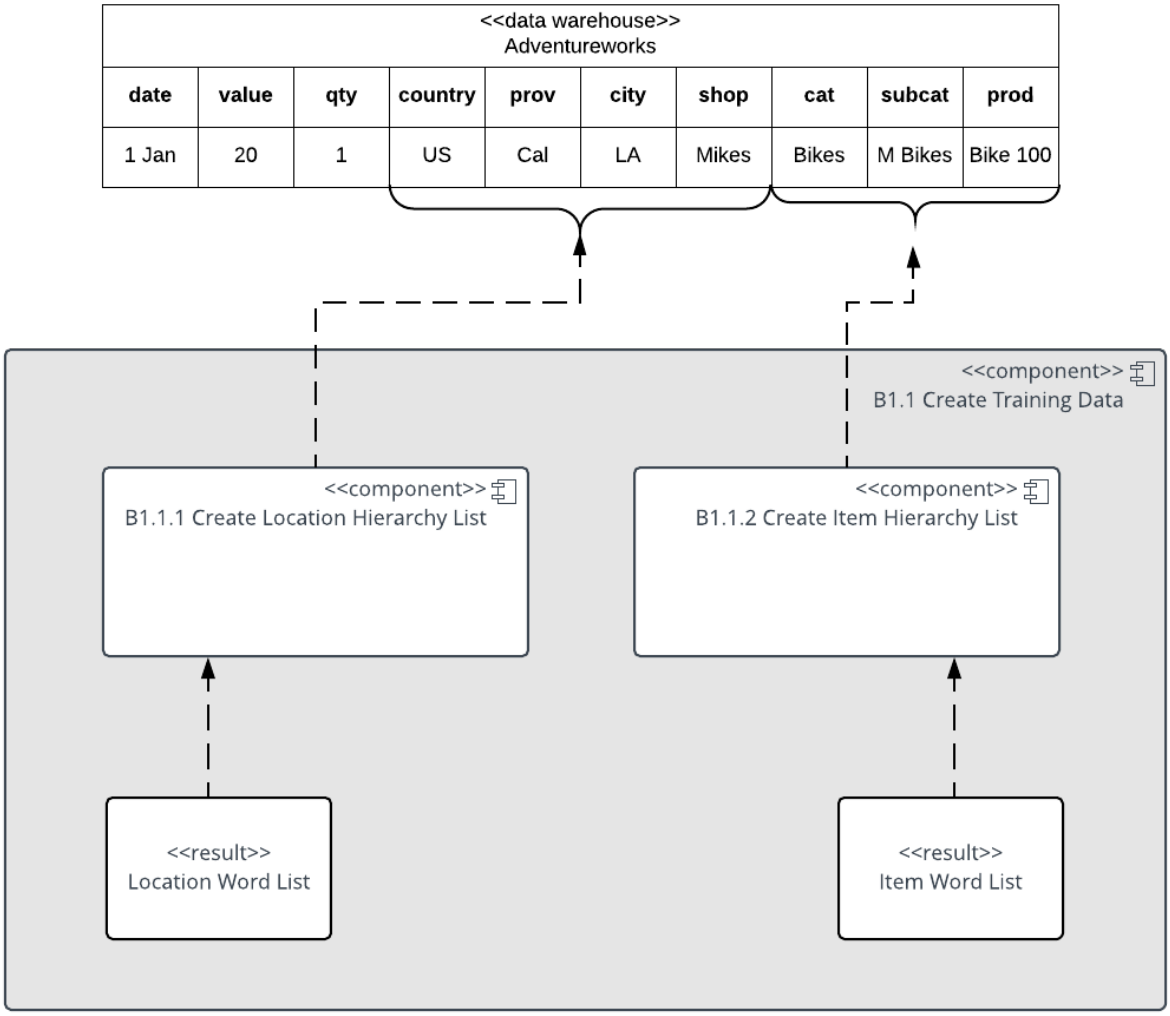

Figure 3.4: A key insight in the research is the level at which word vectors are trained. Once set of word vectors are trained for the location hierarchy related columns, and one set is trained for the item hierarchy related columns.

Table 3.1: The item hierarchy consists of three columns. When the training data for the embedding model is created, the words from the hierarchy are placed in one long string with each row's columns next to each other, followed by the following row's columns, and so forth.

Product	Subcategory	Category
Mountain-100 Silver, 44	Mountain Bikes	Bikes
Long-Sleeve Logo Jersey, M	Jerseys	Clothing

column, resulting in seven models being trained for the one data warehouse. The implication of the first method is that there will only be one vector trained per word for the whole dataset, thus the word 'bike' in the product 'mountain bike' will have the same vector as the word 'bike' in the location 'better bike shop'. The latter approach generates a vector per column per word; consequently, if 'bike' appears in two location columns and three product columns, there will be five different vectors for it.

Another key insight into the research is not to use stop words. Models with and without stop words were compared, and the accuracy of both these model can be seen in the data analysis chapter. Stop words are words such as *in, at, on, the and of*, and refers to the grammatical words instead of the concrete ones, in other words, stop words serve a specific function which does not necessarily carry content or meaning. When comparing average vectors, the effect that stop words have on an utterance is significant. For the utterance "What are the sales for bikes in the United Kingdom?", five ('are', 'the', 'for', 'in' and 'the') vectors have the ability to completely skew the average vector. A typical example would be that the word 'the' would have been trained in the location hierarchy as seen in 'The Mountain Bike Shop'. Thus, when computing the utterance in terms of the location vector, 'the' and 'the' weigh the same as 'United' and 'Kingdom'. Accordingly, stop words were removed in the process of creating training data.

In contrast to these two approaches, a middle ground has been found to yield the best results. This insight dramatically increased accuracy of the model, as can be seen in the data analysis chapter. This middle ground is to generate a word vector per hierarchy in the data warehouse. Hierarchies in data warehouses represent concepts, and in the Adventureworks data warehouse, there are hierarchies for the concepts of location and item. The location hierarchy consists of four columns: country, province, city and store. The item hierarchy is made up of three columns: category, sub-category and product. The training data is then created per hierarchy in the following way: the individual words per column are placed next to each other per row for the three columns. The next row in the data warehouse is then appended in the same fashion. As an example, consider the two rows in the item hierarchy in Table 3.1. This will create the following training data:

$$[mountain, 100, silver, 44, mountain, bikes, bikes, long, sleeve, logo, jersey, m, jerseys, clothing]$$

Following this methodology, one word vector is trained for the word 'bike' relating to the location hierarchy, and one word vector is trained relating to the item hierarchy.

Stemming is a technique that abbreviates words by removing affixes and suffixes. For instance, the word 'loved' and 'loving' both revert to their stem, being 'love'. For this research, stemming was used in earlier iterations of the system, but not applied to the final one. The reason for considering stemming was to cater for variances in words not being trained in the data warehouse but being asked in an utterance. An example is when a user asks for the sales of 'bikes', but only the word 'bike' appeared in the trained vectors from the data warehouse. However, the use of pre-trained global vectors in component *B1.2 Create Embeddings*

negates the necessity for using stemming. The reason for this is because all versions of the word will be included in the global vectors, for example, it is not necessary to stem 'bikes' to 'bike', because the vectors for both 'bike' and 'bikes' are contained in the global vectors, and they are close to each other.

Component B1.2 Create Embeddings

The aim of component *B1.2 Create Embeddings* is to take the training data created in the previous step, and to create word embeddings for each individual word per hierarchy.

A key insight for this component is to use previously trained global vectors. The word2vec algorithm assigns random vectors for each word being trained, but a word might appear so seldom in the data warehouse, that its vector representation might not be accurate, even with a large number of training iterations. Using the global vectors, it gives a starting context to the word. This then begs the question: Why train these words at all, when their global vector has already been trained? The answer is to understand the meaning of the word in relation to others within the specific context of the data warehouse. Even within a data warehouse, the same word can have different meaning, as seen with the 'bike' example for location and item. The 100-dimensional vectors by GloVe, developed by Stanford University, were used in this component.

The skip-gram model is used considering one word to the left, and one word to the right of the target. In this example of training data: [long, sleeve, logo], the algorithm tries to predict long and logo from sleeve.

After training, each individual word per hierarchy has a 100-dimensional vector attached to it. When reducing this to only two dimensions per word by principal component analysis, a graph can be drawn to inspect the model. One would expect similar concepts in the model to appear near each other on the graph.

To demonstrate how training shifts the words in vector space to eventually group similar concepts together, see Table 3.2 and Table 3.3. The first iteration showcases the use without global vectors, to illustrate the movement from pure random allocated vectors to trained vectors. This method is not used in the final methodology, but is a good illustration of the word2vec algorithm. Table 3.2 deals with the Location hierarchy, and it can be seen that the terms nearest to the five chosen words are random; in other words, the German province Saarland has nothing to do with the concept 'east' but, after training, it can be seen that the Saarland is nearest to several German locations, as it appears in the data warehouse. Similarly, the item hierarchy starts off with randomly allocated vectors, as seen by the fact that there are some words (for example, 'bracket') close to the vector for the concept of 58 before training. After training, all the closest vectors are for numbers, see Table 3.3.

The first hyperparameter to choose is the size of the number of vectors to attach to each word. For the final training, a vector size of 100 was chosen, as a good balance between detailed embeddings and long processing times. The next big hyperparameter is the number of epoch to run. For this, 270 001 was chosen. Both the vector size and number of epochs were chosen by inspecting the two dimensional representation of word embeddings (for example in Figure 3.5) and investigating the average loss after each epoch, as shown in Figure 3.7. The last hyperparameter chosen was the skip windows, which is the number of words to consider left and right. For the final training, a skip window of 1 was chosen, because the corpus does not consist of long sentences, where a bigger skip window can give better context to target words. Some of the cells in the data warehouse consists of only a few words, and fits a smaller skip window.

The methodology uses global vectors for initiating the vectors. Table 3.4 displays the closest words initialed by GloVe before and after training for the location hierarchy. When looking at vectors close to 'accessories', it can be seen that everyday examples are listed, for instance, 'equipment'. However, in the

Table 3.2: It can be seen how the nearest words change from random when training the location hierarchy related words without using GloVe.

Location Before Training (No GloVe)
Nearest to saarland: east, howell, work, scooters, milwaukie
Nearest to washington: baskets, next, options, orly, certified
Nearest to shop: exhilarating, efficient, global, retail, contained
Nearest to company: outfitters, accessories, canyon, instant, issaquah
Nearest to england: canada, better, ouen, immediate, stockton
Location After Training (No GloVe)
Nearest to saarland: hamburg, brandenburg, westfalen, hessen, amalgamated
Nearest to washington: colorado, gear, nevada, carolina, tampa
Nearest to shop: emporium, golf, classic, exemplary, suburban
Nearest to company: distributors, orange, escondido, network, motorbikes
Nearest to england: hampshire, carolina, missouri, milton, nevada

Table 3.3: Item related words show that random words appear close to each other before training when not using GloVe for training.

Item Before Training (No GloVe)
Nearest to handlebars: saddles, 30, 54, black, wheel
Nearest to 58: 44, bracket, 38, jerseys, half
Nearest to brakes: 250, oz, accessories, 50, cleaners
Nearest to sleeve: pedals, caps, shorts, 70, crankset
Nearest to wheels: lock, locks, rear, hydration, hl
Item After Training (No GloVe)
Nearest to handlebars: frames, derailleurs, saddles, wheels, brackets
Nearest to 58: 60, 62, 650, 250, 44
Nearest to brakes: wheel, pedals, frames, saddles, derailleurs
Nearest to sleeve: awc, cap, vest, caps, patches
Nearest to wheels: pedals, cranksets, headsets, frames, brackets

Table 3.4: Location related words start out with meaning when training with GloVe, but move to a domain specific meaning of the words after training.

Location Before Training (GloVe)
Nearest to fashionable: trendy, chic, stylish, boutique, casual
Nearest to accessories: leather, toys, hardware, products, merchandise
Nearest to california: texas, florida, oregon, arizona, colorado
Nearest to company: business, corporation, purchase, sales, manufacturing
Nearest to london: sydney, paris, melbourne, york, british
Location After Training (GloVe)
Nearest to fashionable: eastside, chic, preferred, many, elite
Nearest to accessories: wonderful, unique, commendable,preferred,unsurpassed
Nearest to california: utah, daly, missouri, sioux, michigan
Nearest to company: store, lines, distributors, motorbikes, escondido
Nearest to london: keynes, berks, bracknell, reading, woolston

Table 3.5: Related item words start close to each other in vectors space when training with GloVe, but after training has a domain specific meaning.

Item Before training (GloVe)
Nearest to saddles: saddle, handlebars, tights, bikes, shorts
Nearest to wheels: wheel, brakes, rear, tires, frames
Nearest to 3000: 1000, 400, 500, 650, 750
Nearest to water: pumps, wash, bottom, long, full
Nearest to vest: vests, helmet, cap, gloves, helmets
Item After training (GloVe)
Nearest to saddles: wheels, forks, derailleurs, brackets, headsets
Nearest to wheels: brackets, saddles, pedals, headsets, frames
Nearest to 3000: 1000, 50, 54, 2000, 60
Nearest to water: pumps, hydration, half, long, 30
Nearest to vest: vests, jersey, sleeve, helmets, headsets

context of the data warehouse, the word 'accessories' is mentioned in shop names, and thus the vectors are moved to match other vectors in shop names, for example, 'wonderful'.

Each word in the vocabulary used in the location hierarchy has a 100-dimensional vector attached to it after training. By using principal component analysis, this can be reduced to two dimensions. The two-dimensional representation can be seen in Figure 3.5. It can be seen that German-related locations are clustered together in the orange circle.

Similar to the location hierarchy, the item hierarchy initiated by GloVe starts by matching words related in everyday life, such as 'water' with 'wash' but, after training, the concepts moved to data warehouse-related entries, such as 'water' with 'hydration'. Also see Table 3.4 for location words close to each other before and after training with GloVe.

Figure 3.6 shows the two-dimensional representation of the vectors of the item dimension. These vectors are created by principal component analysis of the 100-dimensional vector space to two dimensions. It can be seen that there are indeed patches of similar concepts, for example, colours, types of bikes, clothing or numbers. The orange circle in Figure 3.6 shows the cluster of colour words close to each other, and separate from other concepts such as clothing or numbers. See Table 3.5 for item words close to each other before and after training with GloVe.

The word2vec algorithm is run for 270 000 steps. Figure 3.7 shows the average loss at each step. It can

Figure 3.5: After training, related concepts appear close to each other in vector space. Here, a two-dimensional representation of the location related words can be seen, notice the cluster of German-related words.

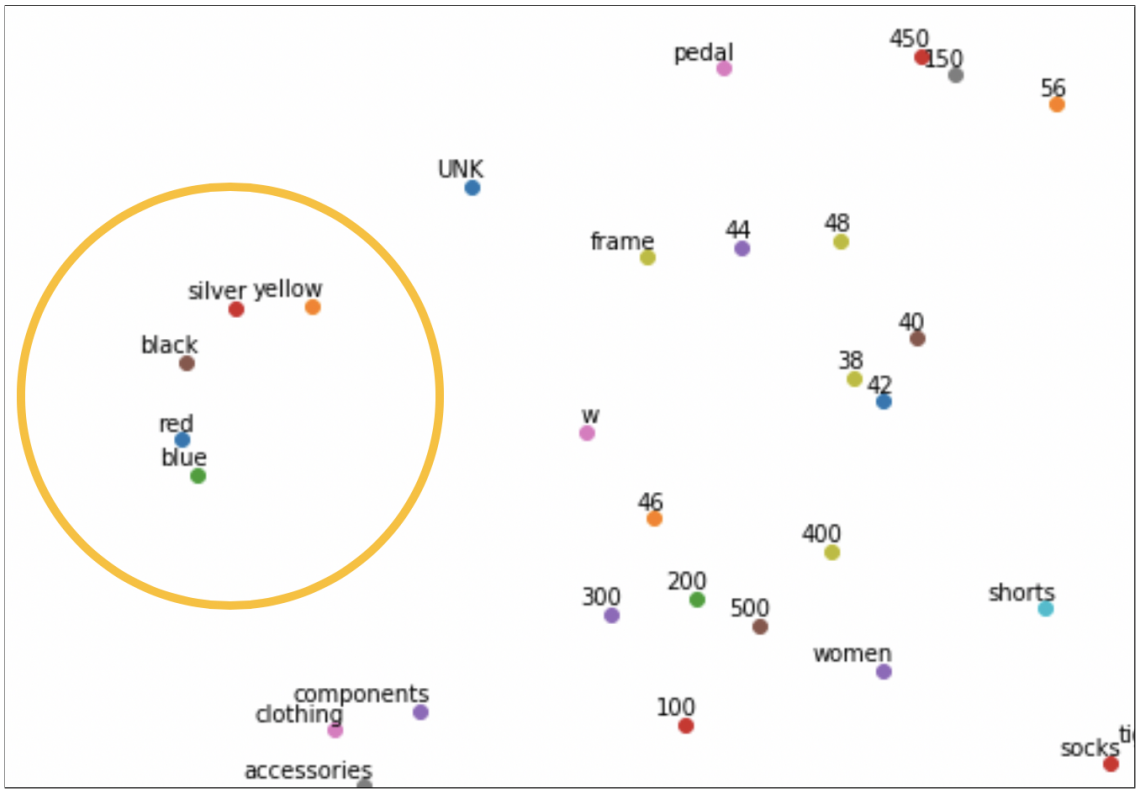

Figure 3.6: After training item related words, the grouping of similar concepts can be seen in a two-dimensional representation of the vector space. See the grouping of colour related concepts.

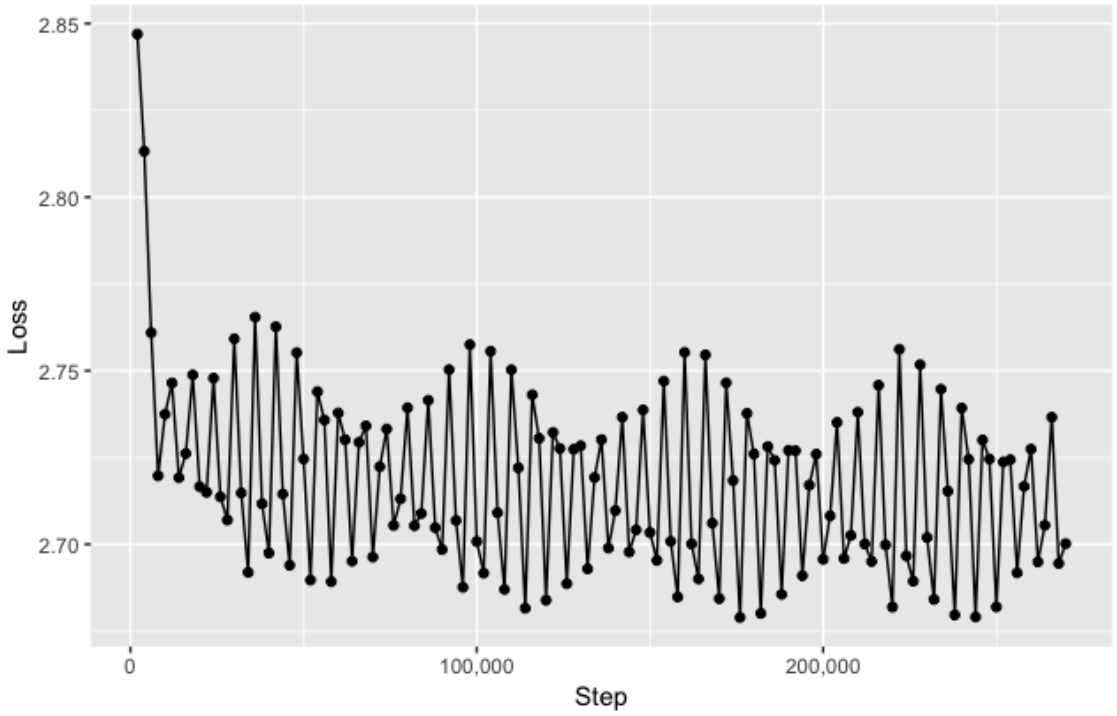

Figure 3.7: The word2vec algorithm completed 270 000 steps. Here displayed is the average loss for the steps.

be seen that the model displays a cyclic pattern and does not improve immensely after about 178 000 steps.

Component B1.3 Create Average Vectors for Columns

Each column in the hierarchy consists of several words per row. The purpose of component *B1.3 Create Average Vectors for Columns* is to calculate average column vectors for each of the columns in the hierarchy by using the word vectors created in components *B1.2 Create Embeddings*.

A key insight into the success of the system is to compare the vector of the full user utterance (utterance vector) to the vector for each unique entry per column (column vector). Each column vector has to be calculated according to the hierarchy to which that word belongs. For example, the column vector for 'mountain bike socks m' would use the item-based word vector for the word 'bike' to calculate the average vector. Likewise, the column vector for 'better bike shop' would use the location-based word vector for the word 'bike'.

Component *B1.3 Create Average Vectors for Columns* is the last component in the training sub-system. Every data artifact has now been created to analyse the user's utterance, and can be persisted in a NoSQL structure, such as MongoDB. Firstly, two sets of word vectors have been trained, one to use in relation to the location hierarchy, one to use for the item hierarchy. Subsequently, each unique value in each column in the data warehouse has an average column vector for comparison with the average vector of the user utterance.

3.4.3 Sub-system B2. Querying

The purpose of the second sub-system is to convert the user utterance to a vector, and then compare that vector to the average column vectors created in sub-system *B1. Training*. After evaluating the matches, the query is executed on the data warehouse, then returned to the user.

Component B2.1 Convert Utterances

Component *B2.1 Convert Utterances* functions similarly to component *B1.3 Create Average Vectors for Columns* in that it created average utterance vectors for text. In the case of component *B1.3 Create Average Vectors for Columns*, it is average vectors for the unique values per column in the data warehouse, whereas component *B2.1 Convert Utterances* converts the user utterance.

Data artifacts from component *B1.2 Create Embeddings* are used to create two average utterance vectors. Firstly, the word vectors created for the location hierarchy are used to create a location-based average utterance vector for the user utterance. Secondly, the word vectors for the item hierarchy are used to create an item-based average utterance vector of the same user utterance. For example, the utterance: "What are the sales for mountain bike socks in Georgia?" would have two average utterance vectors. The location-based utterance vector would be calculated on an average of the location-based words it finds, thus 'bike' and 'Georgia'. In the same way, the item-based utterance vector uses 'mountain' and 'bike' and 'socks' to calculate the item-based average utterance vector.

Component B2.2 Calculate Cosine Similarity

For the query to be constructed, the system needs to know which column to query, and which value to use in the query. In other words, for the system to add "WHERE 'province' LIKE 'Georgia'", it has to get 'Georgia', and understand that Georgia is a province, and not a country or city.

Component *B2.2 Calculate Cosine Similarity* takes the two generated average utterance vectors of component *B2.1 Convert Utterances* and the average column vectors for values in the data warehouse from component *B1.3 Create Average Vectors for Columns* and finds the closest matches. First, the location-based average utterance vector is compared to each average column vector for each column's unique values in the location hierarchy in the data warehouse. The cosine similarity is used to get the nearest column vector in each column. Each comparison with a column in the hierarchy would then bring back an entry and a value. Although the query is constructed with some heuristics in component *B2.3 Construct Query with Heuristics*, it can already be seen that the province column has the highest cosine similarity score, and the value that gets that high similarity is 'Georgia'.

The same procedure takes place to get the best matched item column and value using the item-based average utterance vector of the user utterance, and the average column vectors of the unique values per column in the item hierarchy. If a word is mentioned in an utterance that does not appear anywhere in the data warehouse, that word is simply ignored when calculating the average utterance vector.

Component B2.3 Construct Query with Heuristics

The extraction of the intent (the what) of the utterance and the date component is exactly the same as described in the heuristics model.

The following step in the component is to find the location part of the utterance. For this, the location-related nearest column vectors from component *B2.2 Calculate Cosine Similarity* are evaluated, in other words, the nearest vector for shop, city, province and country. The cosine similarity of each match is adjusted with the following score: the number of words of the closest match is looked up in the utterance, and the score is multiplied by the number of matched words divided by the number of words in the matched value. For example, in the utterance "What are the sales for mountain bike socks in Georgia?" the score for the item match 'mountain bike socks, m' matches three of the four words ('m' is not matched), the cosine similarity score is multiplied by 3/4. The reason for this is to adjust matches that might appear by chance. An example is that the average of 'United' and 'States' might be close to the average of 'South' and 'Australia'. The multiply technique would rule out matches where no words are shared. After the adjustment, the column type and value with the highest score is selected.

Lastly, the item is looked up in the utterance following the same procedure as with location. The nearest item-related column vectors are reviewed, and their scores are also adjusted, with the number of words in the utterance divided by the number of the words in the matched value. The highest item-related column and value is then selected. The procedure is to get the highest score per column, and then step up the hierarchy starting at the lowest level (for example, 'shop' in the location hierarchy), each time selecting the new column when the score is higher or equal to the current highest score. The reason for the 'equal to' part also moving into a new column is to get as big as possible answer, for example, if the user asked for sales in Victoria (in Australia), the province Victoria is selected, rather than the city.

It is now possible to construct a SQL query using the one fact and three dimensions extracted from the utterance.

Component B2.4 Execute Query

Component *B2.4 Execute Query* takes the SQL query generated in component *B2.3 Construct Query with Heuristics* and executes this query on the data warehouse. This will generally bring back one figure (for a question like "What are the sales for mountain bike socks in Georgia?") or a list of values and their figures (for a question like "What are the top cities for the sales of mountain bike socks?"). This result is returned to the user, along with some context. The context consists of the facts that the user asked for, along with their dimensions. In other words, which column did the query bring back, and which columns did it use to filter the query with which values. As an example, if the SQL query looks like: "*SELECT Sum(SalesValue) FROM adventureworks WHERE province LIKE '%Georgia%' AND item like 'mountain bike socks, m' AND date = '2018-07-01'*", then the result will look like this: "The sum of the sales value for the province of Georgia for the item of Mountain Bike Socks, M for 1 July 2018 is: $ 1234.56".

Also built into the result that is returned to the user are all the selected parameters, for example:

{"fact": "sales value",
"item_type": "product",
"item": "mountain bike socks, m",
"location_type": "province",
"location": "Georgia",
"date": "2018-07-01"}

The client (for example, a mobile application) can then cache the values to return in a follow-up question, for example, the user can then follow the mentioned query up with the utterance: "What about short sleeve jerseys?" The system will go through all the same steps in sub-system *B2. Querying*, and only find a match for the item 'short sleeve jerseys'. All the other facts and dimensions will be the same; it will still be in Georgia, but 'mountain bike socks, m' will be swapped out with 'short sleeve jerseys'.

3.4.4 Problems with the Embedding Model

The model must select a location and an item, so does not cater for the 'no_location_selected' or 'no_item_selected' situation. The model also does not handle multiple entries of the same grain, for example, the utterance "What are the sales for mountain bikes in Germany and Australia?" would only yield one selected location. Other limitations are discussed in a subsequent chapter.

Chapter 4

Data Analysis

4.1 Data Analysis Methodologies

To measure the accuracy of the models, the extraction of the facts and dimensions needs to be evaluated. Both the models use the same method for extracting the facts and the date dimension, with the difference between the two methods being the text analysis parts of the data, thus the extraction of the location and item dimensions. To construct the correct query to be sent to the data warehouse, for both the item and location dimensions, the column type and column value need to be extracted from the user utterance. The models are thus compared and evaluated along four accuracy measures, namely location column accuracy, location value accuracy, item column accuracy and, lastly, item value accuracy.

To evaluate the accuracy of the models, random utterances will be generated and run through the different models. To minimise bias introduced by the researcher during the generation of the random utterances, human input has been collected using a survey: the outcomes of system *X1. Human Utterances* influenced the set-up of system *Y1. Random Generated Utterances*.

4.1.1 System X1. Human Utterances

A questionnaire was designed to test users' question style. Responders to the survey were given a snippet of the dataset used for building both models, then tasked to ask questions about the data.

Component X1.1 Profilic Framework

The questionnaire was distributed on Prolific.ac, a paid-for platform used for data collection. For the study, 70 surveys were collected at a cost of GBP 1.22 per submission. Responders can be screened according to a lot of demographics. For this survey, only graduate, full-time employees were asked to participate. The reason for this specific demographic, is because users of a natural language process solution on data warehouses would generally be middle to upper management, fitting the description of graduate full-time employees. In other words, demographics were chosen to align responders as close as possible to real-world users. To demonstrate the survey, one user's responses are documented below.

Timestamp

2018/10/25 7:18:04 PM GMT+2

Please indicate your consent before proceeding:

Value	Quantity	Date	Product	Subcategory	Category	Shop	City	Province	Country
205.146	9	2006-07-01	Full-Finger Gloves, M	Gloves	Clothing	Top Sports Supply	Edmonton	Alberta	Canada
7873.146	9	2006-04-01	Road-450 Red, 58	Road Bikes	Bikes	Retail Mall	Richmond	British Columbia	Canada
71.988	2	2007-04-01	Men's Sports Shorts, M	Shorts	Clothing	Original Bicycle Supply Company	Toronto	Ontario	Canada
418.512	2	2006-09-01	ML Mountain Frame - Black, 38	Mountain Frames	Components	Rapid Bikes	Toronto	Ontario	Canada
2458.9178	2	2007-02-01	Mountain-200 Black, 38	Mountain Bikes	Bikes	Leisure Activities	Hull	Quebec	Canada
76.2	2	2008-05-01	Classic Vest, S	Vests	Clothing	Uttermost Bike Shop	Bracknell	England	United Kingdom
5102.97	5	2007-07-01	Road-350-W Yellow, 40	Road Bikes	Bikes	Prosperous Tours	London	England	United Kingdom
1336.23	3	2007-12-01	Touring-3000 Yellow, 44	Touring Bikes	Bikes	Fun Toys and Bikes	Tucson	Arizona	United States
84.7734	6	2006-12-01	Half-Finger Gloves, M	Gloves	Clothing	Eastside Department Store	Union City	California	United States
1070.694	3	2007-11-01	ML Road Frame-W - Yellow, 38	Road Frames	Components	General Associates	Hollywood	Florida	United States
744.2727	1	2006-07-01	HL Mountain Frame - Silver, 42	Mountain Frames	Components	Valuable Bike Parts Company	Orlando	Florida	United States
3887.964	6	2006-11-01	Mountain-300 Black, 38	Mountain Bikes	Bikes	Noiseless Gear Company	Columbus	Georgia	United States
838.9178	2	2006-02-01	Road-650 Black, 60	Road Bikes	Bikes	District Mall	Ferguson	Missouri	United States
400.104	2	2008-06-01	LL Touring Frame - Yellow, 50	Touring Frames	Components	Sleek Bikes	Denby	South Dakota	United States
35.994	1	2006-11-01	Men's Sports Shorts, S	Shorts	Clothing	Genial Bike Associates	Humble	Texas	United States
392.658	2	2007-02-01	HL Mountain Rear Wheel	Wheels	Components	Genial Bike Associates	Humble	Texas	United States
606.996	3	2006-10-01	LL Road Frame - Red, 48	Road Frames	Components	Friendly Bike Shop	Bellingham	Washington	United States
377.946	9	2008-06-01	Women's Mountain Shorts, L	Shorts	Clothing	Metro Cycle Shop	Tacoma	Washington	United States

Figure 4.1: A sample of the Adventureworks data were given to respondents of the survey in order to complete the questions.

I consent, begin the study

Please enter your Prolific ID:

5aa1c9fe35237b000112df7d

What is the highest level of education you have completed?

Graduate degree (MA/MSc/MPhil/other)

What is your employment status?

Full-Time

Component X1.2 Google Survey

Users were given the random rows of the Adventureworks dataset as displayed in Figure 4.1. The questionnaire consisted of two main sections; firstly, in the general section, responders were tasked to write an open-ended question on anything in the data warehouse. Then, in the specific section, users were asked to give three different variations of a specific question.

In the general section, after the responders supplied the question, they were tasked with identifying the different components related to their own query. Here is an example of a response:

1. *Write down a question you might ask from the data.*
   How many bikes were sold in Canada during 2006?
2. *Which column contains the answer to your question?*
   Quantity
3. *Identify the part of your question pertaining to the sales value or quantity.*
   How many bikes
4. *Identify the part of your question pertaining to the date.*
   during 2006
5. *Identify the part of your question pertaining to the product.*
   bikes
6. *Which product related column does your question refer to?*
   Category

7. *Identify the part of your question pertaining to the location.*
   Canada
8. *Which location related column does your question refer to?*
   Country

This data can be used to identify the columns and the value of the columns to look up, in other words, accuracy of the models can be tested using this response. The quality of the responses for the general section varied, in that some responders missed identifying components of their own question.

The responses for section two, the specific question section, were better. The responder was tasked to put the following question in three different ways: "What is the sales quantity for shorts in Ontaria between 21 November 2018 and 28 November 2018?" A responder's answers can be seen below.

1. *First version of the question you might ask from the data.*
   How many shorts were sold between November 21st and 28th in Ontaria?
2. *Second version of the question you might ask from the data.*
   What was the total amount of shorts sold in Ontario between 21 November and 28 November?
3. *Third version of the question you might ask from the data.*
   Between the 21st and 28th of November, how many shorts were sold in Ontario?

Component X1.3 Survey Results

When trying to understand the user's question, the system needs to firstly understand *what* to bring back, and then *how* that should be filtered. The *what* can be equated to the intent of the utterance, and the *how* is similar to the entities on which to filter the query. From the survey, the *what* is identified by question *2: Which column contains the answer to your question?* When either value or quantity is returned, the sales intent is triggered. The results of the survey indicate that 41.4% of the respondents chose the sales value and 28.6% chose the sales quantity. The next survey sections dealt with how the data should be filtered. The first question to address the filtering is question *6. Which product related column does your question refer to?* The results include that Product scored 54.3% of the answers, while category and sub-category each scored 17.1%. The results also indicated that some respondents did not understand the question, because some listed Country as a column by which the product should be filtered.

Similarly, the location question is: *8. Which location related column does your question refer to?* The four location-based hierarchy columns once again make out the bulk of answers, with Country at 32.9%, City with 21.4%, Province with 18.6% and Shop with 11.4%. Other questions in this section were text-based, and did not reveal any interesting data.

Even though the second section (specific question style) of the survey was also only text-based, some text analysis was done to understand question style. Visual inspection of the responses showed that it was apparent that the sales intent is addressed in a variety of different ways. Analysing this further yields that 52.8% of respondents used the word 'sold', 34.4% used 'sales' and 12.8% used 'sell'. It is clear that not only the word 'sales' is used to trigger the sales intent, but also 'sell' and 'sold'.

Component X1.4 Survey Influence on Random Generated Utterances

To identify the sales intent, both algorithms looked for the keywords 'sales', 'sell' and 'sold', as would be asked in the question: "What are the sales for bikes in Canada?" One of the insights from the survey, is that the keyword 'sold' also needs to trigger the same intent, as mentioned in the same but differently-worded question: "How many bikes sold in Canada?"

4.1.2 System Y1. Random Generated Utterances

The first method of testing the accuracy of both models involves generating user utterances, and then evaluating the results.

Component Y1.1 Generate Random Utterances

An algorithm to mimic the way users would interact with the data warehouse is used to create random user utterances. It generates a sales fact randomly from heuristics, opting for either sales quantity, sales value or top sales. A random date (or dates) is then chosen. This can take the form of actual dates in different formats, for example, '1 January 2018', or words referring to a date range, for example, 'last week'.

The algorithm will choose a random column in the location hierarchy and choose a random value in that column. In the same way, a random column in the item hierarchy is chosen, along with a random value from that column.

Two factors of human usage are incorporated. Firstly, humans might not remember every single word from a value in the data warehouse, for example, a user might ask for 'mountain bike socks' when such a product does not exist word for word in the data warehouse (where the entry is 'mountain bike socks, m'). To mimic this behaviour, the algorithm randomly leaves out a word when the total amount of words in the chosen value is more than five words. Another human behaviour is to switch the order in which the words for a product description, for instance, appear in the data warehouse. An example of this is a human calling the item 'red short sleeve shirt', when it is in fact listed as 'short sleeve shirt, red' in the data warehouse. To cater for this, the algorithm switched the word order to random when there are more than three words in the chosen value.

Combining these four randomly chosen artifacts, a user utterance is created. An object containing the generated utterance, along with the answers to the slots to be filled is generated. This utterance can then be tested with the different algorithms, and its accuracy evaluated.

The correct answers are also listed with the utterance, in order to evaluate the models. Notice that the order of the item has been shuffled to mimic human speech. See two of the randomly created utterances:

{'return_type': 'sales',
'return_column': 'value',
'date_begin': '2018-01-13',
'date_end': '2018-01-13',
'location_column': 'province',
'location': 'victoria',
'item_column': 'product',
'item': 'mountain 200 silver 42',

‘utterance’: ‘how many silver mountain 200 42 were sold at victoria on 2018/01/13 ’},
{‘return_type’: ‘sales’,
‘return_column’: ‘quantity’,
‘date_begin’: ‘2018-11-17’,
‘date_end’: ‘2018-11-17’,
‘location_column’: ‘shop’,
‘location’: ‘village tours’,
‘item_column’: ‘product’,
‘item’: ‘ml road frame red 48’,
‘utterance’: ‘what is the sales quantity for the last day at village tours for red ml road 48 frame’},

The utterances created with the words ‘sold’ or ‘sell’ instead of ‘sales’ can be noticed. Here are more examples, to demonstrate the variety of random utterances:

- how many silver mountain 200 42 were sold at victoria on 2018/01/13
- what is the sales quantity for the last day at village tours for red ml road 48 frame
- what is the top province on 2018-04-20 at united kingdom for gloves half finger m
- how many red road frame 62 ll were sold at south australia on 2018/01/13
- how many men bib s shorts l were sold at france between 1 january 2018 and 7 january 2018
- what is the sales quantity for yesterday at greeley for 54 1000 touring blue
- what is the top city on 2018-04-20 at braintree for pedals
- what is the top shop from 2018-07-01 to 2018-07-08 at community department stores for ll road 60 black frame
- what is the top city from 2018-07-01 to 2018-07-08 at united states for handlebars
- how much accessories did we sell at new mexico for yesterday
- what is the sales value for the last day at new hampshire for tights
- what is the top country for yesterday at germany for clothing
- what is the top sub_category on 2018-04-20 at united states for mountain frames
- what is the sales quantity for last week at united states for accessories
- what is the sales quantity on 2018/01/13 at australia for brakes

Component Y1.2 Evaluate Random Utterances with Both Models

One thousand randomly generated utterances from component *Y1.1 Generate Random Utterances* were then tested according to the heuristics and the embedding model, noting the four columns on which the two models differ: location column, location value, item column and item value.

Table 4.1: The results of different models are compared on 1000 random utterances, with two models chosen for further comparison; one heuristics model and one embedding model.

Model Type	**Heuristics**		**Embedding**				
Notes	**Chosen Heuristics Model**	The test results for the heuristics model on perfect questions, there were no Shuffle and Skip in this model, and the results were thus discarded	**Chosen Embedding Model**	No GloVe vectors in this model in order to demonstrate the absence of a key insight, and the model was thus discarded	Stop Words included in this model in order to demonstrate the absence of a key insight, and the model was thus discarded	Only one set of word vectors in this model in order to demonstrate the absence of a key insight, and the model was thus discarded	Model to demonstrate the failed attempt when using Max Pooling, and the model was thus discarded
return_type	**1000**	1000	**1000**	1000	1000	1000	1000
return_column	**1000**	1000	**1000**	1000	1000	1000	1000
date_begin	**999**	999	**999**	999	999	999	999
date_end	**999**	999	**999**	999	999	999	999
location_column	**964**	992	**988**	987	977	899	731
location_score	**967**	996	**982**	980	969	837	680
item_column	**727**	1000	**995**	994	997	867	945
item_score	**726**	1000	**978**	976	980	820	923
Total	***701***	*992*	***956***	*955*	*945*	*680*	*603*

4.2 Results

To test the accuracy and speed of the models, 1 000 random utterances were generated and passed through both models.

Table 4.1 compares the results between the two chosen models, and other iterations that display the effects of attempts of different changes to the models. The first four results (return_type, return_column, date_begin and date_end) are shared between the algorithms, as they all use exactly the same rules to determine these values. Thus, to compare the models with each other, only the last four results are compared (location_column, location_score, item_column and item_score).

4.2.1 Models Not Used

Several models were analysed, for instance, to indicate the effect of key insights or to showcase a failed attempt at increasing accuracy.

Heuristics: No Shuffle and Skip

The first model that was analysed, but not chosen, is a version of the heuristics model that was tested on perfect questions; in other words it does not mimic the way humans construct questions. This means that none of words in the randomly-generated utterances were shuffled or skipped. This approach would, in theory, bring back perfect results, and in practice had an overall 99.2% accuracy. The biggest contributing factor to missed matches is the missed location column detection, with an accuracy of 99.2% and the reason for this is that there are provinces and cities with the same name, for instance, Hamburg appears as a province and a city. Thus, the randomly generated utterance will be marked as the city, and the heuristics model will detect it at the highest level of this hierarchy, being the province. This model was not chosen for the final heuristics model, because the questions were too perfect, and did not represent how humans speak.

Embedding: No GloVe

One of the key insights was to initiate the word vectors with global vectors using GloVe. To illustrate the effect that this has on the accuracy, an embedding model without global vectors were tested. This model included all other key insights, except the global vectors. The accuracy is very similar to the final chosen model, with the final model on 95.6% accuracy and the no-GloVe model with 95.5% accuracy. This is not a big improvement on accuracy, but it is still marked as a key insight due to the way in which the random utterances were set up. The random utterances never exchanged word synonyms, which would be a strong case for the use of global vectors. Consider the following example: the user asks, “What are the sales of jerseys at the Good Bike Shop?” The user got the name of the bike shop wrong, as there is no shop in the data warehouse called the Good Bike Shop. What the user referred to, was the Great Bike Shop. Because the word vectors were initiated with global vectors, the word vector for good is very close to the word vector for great, and thus, the average of the ‘Good Bike Shop’ is close enough to pick up the ‘Great Bike Shop’. This would not have been possible with a randomly-instantiated word vector if the word ‘Great’ was not prevalent in the data warehouse.

Embedding with Stop Words

Another key insight is that stop words can skew the average column or utterance vectors to such an effect that the accuracy is 94.5% compared to the chosen model where stop words have been left out, where the accuracy is 95.6%. This model only added stop words and thus included all other key insights. The reason for this is that stop words appear in unrelated columns in the text, thus a stop word is not linked to a concept. The second reason is that the utterance can contain a lot of stop words compared to the number of non stop words, for example, the utterance "What are the for bikes in the United States?" contains more stop words than non-stop words, thus skewing the average vector.

Embedding: One Set of vectors

The key insight with the biggest increase in accuracy was to train different sets of column vectors for the location and the item dimensions. The model trained on only one set of column vectors for both dimensions resulted in an accuracy score of 68%. This provides a good insight into the workings of embeddings, meaning that the reason for creating embeddings in general is to give meaning to concepts. In other words, the word 'bike' in the data warehouse represents a concept that can be related to other concepts in the data warehouse by calculating distances between the vectors. Thus, training different vectors for the word 'bike' from a location point of view compared to the item point of view makes sense, because they are indeed two different concepts. From a location point of view, 'bike' can be used to describe a shop's name, for example, 'The Better Bike Shop', and could appear close in vector space to another word that describes a shop, for example, 'bargain'. From an item point of view, 'bike' represents the item with the pedals that is sold, for example, 'mountain bike' and could appear close in vector space to an item that is also sold, for example, 'handles'.

Embedding: Max Pooling

The calculation of the column vectors and the utterance vectors happens by merely taking the average of the word vectors used in either the column or the utterance. One method that was tried to improve on the averaging on vectors stems from max pooling which is a technique used on convolutional neural networks. Instead of just taking the average of the mentioned vectors, a concept similar to max pooling was applied. So, when combining two vectors, instead of taking the average between each of the 100 values to create one new set of 100 values, the maximum of each of the dimensions was taken. For the model, all other key insights were included, and average vectors were replaced with maximum vectors. Thus, when combining two vectors, for each value, the maximum of the two is chosen in the new vector. The accuracy of the item and location-related results decreased dramatically, from 95.6% in the chosen embedding model to 60.3% accuracy. The technique was discarded. A possible explanation for this result is that the max pooling technique might work better with a higher number of dimensions but, when combining 10 utterance words, the resulting combined maximum vector does not resemble any vector whatsoever in the data warehouse.

4.2.2 Chosen Models

For the heuristics model, the shuffle and skip version was chosen to represent a solution that can work on human-like utterances. The embedding model included all the key insights, and did not use the max pooling technique for combining vectors.

Table 4.2: The speed in seconds of analysing 1000 random utterances.

	Heuristics	Embedding
Total time	16.858989 s	1.654887 s

Accuracy

Comparing the two bold columns in Table 4.1 reveals that the embedding model is more accurate in each of the compared results. The final result is calculated on individual utterances that were completely mapped correctly. It is seen that the heuristics algorithm is 70.1% accurate and the embedding model is 95.6% accurate.

Speed

Table 4.2 shows the amount of time it takes to process 1 000 queries. It can be seen that the embedding model is more than 10 times as quick as the heuristics model. Even though the speed of the heuristics model is not bad for the small dataset in this research, bigger data sets might slow down significantly. To parse one utterance with the embedding model takes just 0.00165 seconds. The speed test was run on a MacBook Pro 2017 with a 2.3GHz Intel Core i5 processor and 8GB 2133 MHz LPDDR3 of memory.

To understand what type of errors each model makes, the individual errors were examined.

4.2.3 Errors for Shared Columns

All the values in the return_type results (for example, 'sales') and the return_column (for instance, 'value') were mapped correctly. In the date_begin and date_end results, there were one error each, caused by the same query. The dates were given as 1 January 2018 to 7 January 2018, but the location for the query was 'weekend bike tours', which contains the word 'week'. This caused the date part of both algorithms to consider the time period as last week. This can be fixed in a following iteration by switching the order of the regular expression looking for dates first before the date-related keywords are identified.

Utterance: what is the sales value between 1 january 2018 and 7 january 2018 at weekend bike tours for mountain wheel rear hl
Correct Answer: 2018-01-01
Wrong Parse: 2018-11-05
Utterance: what is the sales value between 1 january 2018 and 7 january 2018 at weekend bike tours for mountain wheel rear hl
Correct Answer: 2018-01-07
Wrong Parse: 2018-11-11

4.2.4 Errors for Different Model Columns

The first four results were the same across all models (return_type, return_column, date_begin and date_end). Thus, the last four results were used as a comparison between different models (location_column, location_score, item_column and item_score).

Location Heuristics Errors

There were 36 errors for the location column and 33 for the location value. All the errors fall into one of two types:

Type: Shuffle To cater for human speech which switches the order of words, the random utterances shuffled word order when either the location or the items consisted of more than three words. The shuffle is difficult for the heuristics model to pick up, as it looks for the longest consecutive number of words in the utterance at the highest level of the hierarchy. When the word order is shuffled sufficiently, the heuristics model will not pick it up at all.

Utterance: how much ml front wheel road did we sell at adults and cycle kids shop on 2018/01/13 Correct Answer: kids and adults cycle shop Wrong Parse: no_location_selected

Type: Same city and province The heuristics algorithm matches the longest sequence of words in the utterance at the highest level of the hierarchy. In the data, there is a city called Hamburg and a province called Hamburg. Other ambiguous city/province combinations include New York, Victoria, Quebec and Ontario. When a random utterance is created that selects one of these values from the city column instead of the province column, the algorithm will match the value to the province column.

Utterance: what is the top country for the last day at hamburg for road frames Correct Answer: city Wrong Parse: province

Item Heuristics Errors

There were two error types for the 273 errors for the item column and the 274 errors for the item value. Both types of errors are related to the shuffle, and the reason for the high number of errors for the item, as compared to the location errors for the heuristics algorithm, is because item descriptions consist on average of a bigger number of words. Thus, the shuffle and skip are much more prevalent in the item utterance generation.

Type: Shuffle (no_item_selected) The shuffle causes the model to not pick up anything from the item hierarchy.

Utterance: what is the sales value on 2018/01/13 at seine et marne for touring 46 hl frame yellow Correct Answer: hl touring frame yellow 46 Wrong Parse: no_item_selected

Type: Shuffle (Wrong Item Selected) The shuffle does not cause the no_item_selected error; however, the wrong level of the hierarchy was picked up. The user wanted sales for the item 'men s bib shorts l', but the algorithm returned the category 'shorts' in general.

Utterance: what is the top country on 2018/01/13 at racine for men l bib s shorts Correct Answer: men s bib shorts l Wrong Parse: shorts

Location Embedding Errors

The embedding algorithm returned just 22 errors in total; 12 errors for the location column and 18 errors for the location value (with 8 shared errors). These errors can be classified into one of four types.

Type: Same City and Province The first type of error is the same as seen in the heuristics model for returning the correct value, but at the wrong level. The embedding algorithm functions the same as the heuristics algorithm in that it brings back the best matching value at the highest level of the hierarchy. This is achieved by checking the highest cosine similarity score and moving up a level if the next cosine similarity is higher or equal to the current highest cosine similarity. The random utterance in this instance has chosen Hamburg at city level, but the algorithm mapped it to the province.

Utterance: what is the top country for the last day at hamburg for road frames Correct Answer: city Wrong Parse: province

Type: Big Vector Overlap and Item Name Skewing Location The error is caused firstly by the fact that the average vectors for the two shops are very close to each other. However, the factor that skews the average vector towards the 'red bicycle company', instead of the 'black bicycle company' is the item which contains the word red: red road 58 650. Thus, the embedding model got the average of these words from the utterance: 'black bicycle company red road', and matched it to the closest vector in the location hierarchy. In this instance the 'red bicycle company' vector was closer to the utterance than the 'black bicycle company'.

Utterance: what is the sales value for the last day at black bicycle company for red road 58 650 Correct Answer: black bicycle company Wrong Parse: red bicycle company

For errors where vectors were skewed, it is important to note that the correct answer was somewhere in the results, however, not in the top position as selected by the algorithm. In the example question mentioned above, 'black bicycle company' was the correct answer, but 'red bicycle company' was selected. When inspecting the actual results in terms of their cosine similarity, it can be seen that the correct answer ('black bicycle company') was returned in the fourth place:

['red bicycle company', 0.4242149, 'reasonable bicycle sales', 0.42035675, 'original bicycle supply company', 0.41482708, 'black bicycle company', 0.40820593, 'historic bicycle sales', 0.40789863]

This can be used in a future version where, either the user gets prompted when cosine similarity scores are close to each other, or the algorithm looks for matching words in the top ten results returned.

Type: Item Name Skewing Location This is a similar type of vector skewing as seen in the error described above. In this instance however, the correct answer and wrongly parsed suggestion do not overlap at all. It can be seen from the utterance that the letter 'w' for the item name played a bigger role than the word 'zeeland' in matching the average of the utterance to something in the location hierarchy.

Utterance: what is the top product on 15 march 2018 at zeeland for yellow w 550 road 42 Correct Answer: zeeland Wrong Parse: w york

Type: No Vector Overlap The last type of error is a result of working with average vectors. Even though the exact word 'bothell' exists at the shop level in the hierarchy, all other words in the utterance skewered the match away from 'bothell', to land at a position which is closest to the 'united kingdom', even though the words 'united' or 'kingdom' are not even mentioned in the original utterance. This issue can also be addressed by implementing a multiplier for overlapping words.

Utterance: how many 44 3000 touring blue were sold at bothell from 2018-07-01 to 2018-07-08
Correct Answer: city
Wrong Parse: country
Utterance: how many 44 3000 touring blue were sold at bothell from 2018-07-01 to 2018-07-08
Correct Answer: bothell
Wrong Parse: united kingdom

Item Embedding Errors

Similar to the location errors from the embedding model, there were only 22 item related errors in total from the embedding algorithm: 4 for item column and 22 for the item value (with 4 shared errors).

Type: Big Vector Overlap This is an error comparable to the vector overlap error seen in the location errors. In this case, the letter 's' played the role in skewing the results towards the 'men s sports shorts s' item instead of the 'men s sports shorts m' item. This item is created by deleting the apostrophe from 'men's' to 'men s', which caused the error.

Utterance: how much m men s sports shorts did we sell at augusta on 15 march 2018
Correct Answer: men s sports shorts m
Wrong Parse: men s sports shorts s

Type: Location Name Skewing Item In this error, the location (racing supply distributors) skews the average vector towards 'racing socks m' rather than the correct 'tights'.

Utterance: what is the sales quantity on 2018/01/13 at racing supply distributors for tights
Correct Answer: tights
Wrong Parse: racing socks m

Chapter 5

Discussion

The purpose of this research was to develop and compare two different methods for using natural language to interact with a data warehouse. The utterance is a sentence by a user which has to be interpreted by both algorithms. In order to be able to create a SQL query, the utterance firstly had to be analysed for an intent, for instance, does the user want the sales value or the top items sold at a certain place? Then the filter conditions had to be extracted from the utterance; in other words, how the query will be filtered. The query can be filtered according to one of three dimensions in the data warehouse, the date, the location and the item.

For the intent identification and the date filtering, both algorithms used the same techniques.

5.1 Similarity Between Two Chosen Models

The method employed to identify the intent relied on keyword identification using regular expressions. Both models cater for the two types for intents, namely, the returning of the facts in the data warehouse (sales value or sales quantity) or the return of a column in one of the dimension hierarchies ranked by the fact, for example, the top category in a country. The results from the random utterances indicate that the keyword identification performed well, parsing all the return types and return columns correctly. The selection of keywords was adapted from the results of the survey to understand question style. Before the survey, the code only catered for looking for the word 'sales' but the results from the survey showed that users tend to pose the question in different ways, using the words 'sold' or 'sell'.

The fact that the date identification part of both models did not achieve 100% match rate was determined by the fact that a store name contained a keyword looked for by the date identification part of the algorithms.

5.2 Comparison of Two Chosen Models

Analysis on the random utterances showed that the embedding algorithm performs much better than the heuristics algorithm using both metrics of speed and accuracy.

5.2.1 Speed

Addressing the speed of the comparison, it is clear that the embedding model is computationally much heavier than the embedding model. Taking a simple utterance like "What is the sales quantity for last week

at United States for accessories?", by the time the location analysis happens, this sequence has to be analysed: "What is the for at United States for accessories". The intent and date component of the utterance has been removed. Stop words cannot be removed from the utterance, as some location and item names contain them, and would break the consecutive match if removed, thus all the sequences in Table 5.1 have to be matched against every unique entry in every location-related column in the location hierarchy for the heuristics model.

The location match is stripped from the utterance, and the procedure is repeated for all the entries in the item hierarchy. This method can be optimised in that anything below a one-word match which does not match does not have to be tested, for example, if nothing in the hierarchy matches the word 'at', do not test 'at united' or beyond.

In contrast to the heuristics algorithm, looking at the location match, the embedding algorithm creates two average vectors of the utterance, and does one cosine similarity comparison with each of the unique entries in the location hierarchy. This is also repeated for the item component of the utterance. The speed test results were done on a MacBook Pro, but graphics processing unit instances are optimised for linear algebra, which would speed up the results even more.

Another important factor that has to be considered when discussing the speed test results is that the heuristics algorithm cannot compute the location and item components in parallel, because the location component of the utterance has to be removed before evaluating the item component, to save on comparison iterations. For the speed test done on this research, the embedding algorithm did not compute the item and location components in parallel; however, the method lends itself to parallelisation. Thus, when this method is applied on bigger datasets, the item and location evaluation components can be split, run in parallel and the results combined, which is not possible with the heuristics algorithm.

This shows that although the embedding algorithm was already ten times faster than the heuristics algorithm, the embedding model still has the potential to be sped up dramatically.

5.2.2 Accuracy

The main difference in accuracy for the two methods can be explained by the inability of the heuristics model to handle the shuffle and skip of words from the random utterances. This does mimic how humans interact, and is part of developing a robust model that can handle real-life utterances.

The difference in philosophy for the two models is that the heuristics model tried to match words from an utterance with words in a data warehouse, and the embedding model tries to match a concept from an utterance with a concept in a data warehouse. For the embedding algorithm, the word order does not matter; 'red socks medium', is exactly the same thing as 'medium red socks'. This is the fundamental difference between the two approaches, and the reason why the word2vec algorithm was applied; to turn a collection of words into concepts.

5.3 Embedding Key Insights

The embedding algorithm is the better performing of the two compared algorithms. Four key insights made the method possible, firstly, matching the average vector of the utterance to the average vector of columns, then grouping concepts together for training; thirdly, removing stop words and, lastly, using global vectors.

The main key insights were to create column vectors for the unique values in each column in the data warehouse, then creating average utterance vectors for the whole utterance, then to match the nearest column

Table 5.1: All the combinations of words to be tested for the heuristics model.

what
what is
what is the
what is the for
what is the for at
what is the for at united
what is the for at united states
what is the for at united states for
what is the for at united states for accessories
is
is the
is the for
is the for at
is the for at united
is the for at united states
is the for at united states for
is the for at united states for accessories
the
the for
the for at
the for at united
the for at united states
the for at united states for
the for at united states for accessories
for
for at
for at united
for at united states
for at united states for
for at united states for accessories
at
at united
at united states
at united states for
at united states for accessories
united
united states
united states for
united states for accessories
states
states for
states for accessories
for
for accessories
accessories

value to the utterance. The initial thinking was that there is not a lot of overlap between item-related columns and location-related columns, thus in an utterance such as "What are the sales for socks in Germany?" from the full utterance, only the vector for 'socks' would be brought back from an item perspective and only the vector for 'Germany' from a location perspective would be brought back. The data showed that there is indeed a degree of overlap, for example, the word 'bike' appears in both the item and location hierarchy. This was then handled by the next key insight.

The second key insight was to train one set of word vectors for the full item hierarchy, and train one set of word vectors for the full location hierarchy. The word 'bike' would then have different vectors in the context of either location or item.

The third key insight was to remove all stop words from the utterances or item and location names before calculating average vectors. The reason for this is that a stop word like 'for' can skew average vectors to a large degree. Consider the following example: "What are the sales for yesterday for jerseys for Australia?" The word 'for' appears three times and would skew the average vector for the utterance to a degree that jerseys or Australia would not be matched by the cosine similarity.

The last key insight was to use Global Vectors to initiate the word vectors instead of random initiation. The main idea of the embedding method is to train concepts related to this specific domain: a bike chain selling several products across a wide geographical area. If specific domain knowledge did not matter, then the use of only GloVe vectors could have worked, but the word 'bike' in this context means something very specific in terms of the item and location for the Adventureworks data. The usage of GloVe helped with the initialisation of the vectors for the word2vec algorithm, meaning that some of the original meaning of the words still remain and are useful. This is related to transfer learning, which means that previously-trained models are used, the last layer in the neural network is cut off, then trained on a new dataset.

5.4 Failed Attempts

Several noteworthy attempts to improve the embedding model failed. The biggest attempt was to try to improve on the average vectors for combining column values and utterances. The max pooling technique was tried, but yielded poor accuracy results. Accuracy on max pooling was 60.3% compared to the chosen embedding model with average vectors, which yielded 95.6% accuracy.

The grain at which to train the word vectors to use for creating average column vectors was a point which required many attempts. The first attempt was to just throw all words in the one big word cloud and train one set of vectors. In other words, all words in one row of data would appear next to each other to be fed into the word2vec algorithm. The result of this was that overlapping words between the item and location hierarchies only had one vector associated with it, even though these two hierarchies represented two different concepts for the same word. For instance, the word 'bike' had the same vector which applied to mountain bikes (the item) and The Better Bike Shop (the location). This created a lot of conflict when trying to analyse an utterance that contained overlapping words. Next was to train vectors for each column in each hierarchy on its own. Sparse data in high levels of the hierarchy caused insufficient training data and no meaningful vectors were created to be useful for analysis. The best grain was to train one set of word vectors for anything related to the item hierarchy and one set for anything related to the location hierarchy. This matches the existence of the word2vec algorithm in general, as vectors were trained on concepts, such as location or item.

5.5 Next in Workflow

This research ends with the identification of intent and entities in a user utterance. As described in the methodologies, this is only the first, albeit the most difficult sub-system, of a full chatbot on a data warehouse. The next step would be to take the identified intents and entities and convert them to a SQL query. The intent governs which type of SQL *SELECT* clause the query would have by using the return_value and return_type results from parsing the utterance.

The sales intent would create the following type of query: *SELECT SUM(value) FROM fact_sales WHERE ...* On the other hand, the top intent would create a query like: *SELECT country, SUM(value) FROM fact_sales WHERE ... GROUP BY country ORDER BY SUM(value) DESC.*

In each case the WHERE clause is populated by the entities, using the date_begin, date_end, location_column, location, item_column and item results from parsing the user utterance. As an example, the WHERE clause could look like this: *WHERE country = 'australia' and date BETWEEN '2018-01-01' and '2018-01-07'.*

Application Programming Interfaces can be built on top of the two methods, firstly to parse the query and secondly to create and run the SQL. Once this has been done, the solution can be deployed to any user interface, from a web solution to a mobile application.

5.6 Limitations

One of the biggest limitations of the study is the size of the training data. Although the data display many of the difficulties with working with chatbots, such as ambiguity of words, the sheer size is relatively small. Sales data warehouses can house more than 120 000 individual products, whilst the Adventureworks dataset only had about 250.

A limitation in both methods is the use of regular expression to detect the intent. The detection of the date with regular expressions is permissible, because the number of ways a date can be referenced is ideally suited for regular expressions. However, an alternative way to identify intent has to be found to make the whole solution more robust.

Another limitation of the research is the fact that word2vec vectors are not trained to be aggregated. Models like the universal paraphrastic sentence embeddings model could specifically be trained to yield embedding that can be aggregated (Wieting et al., 2015).

The use of synthetic data to test the models is also a limitation in the research. Test data was created by an algorithm, but the ideal way to test models would be to only use real human evaluations.

Chapter 6

Conclusions

The main aim of the research was to develop and compare two algorithms for using natural language to interact with a data warehouse. The data used for the research was the open source Adventureworks sales data warehouse by Microsoft on a fictional bicycle company. The difficulty with interacting with a data warehouse using natural language is to use the user's utterance to identify the user's intent and entities. Once the intent and entities have been identified by the algorithm, the easy part is to construct a SQL query. The data warehouse consists of the sales facts in the middle of the star schema, surrounded by the three classic dimensions: date, location and item. Both the location and the item dimensions consist of hierarchies which span several columns in the data warehouse.

Two methodologies were compared to identify the user's intent and entities, of which the first was the heuristics algorithm. The main aim of the algorithm was to identify the longest sequence of words which matches the highest level of the hierarchy in the data warehouse. The algorithm achieved this by looping though an increasing number of words in the utterance and comparing it with each unique value in each column in the data warehouse.

The second algorithm was called the embedding algorithm and is based on the word2vec algorithm developed by Mikolov et al. (2013). The premise of the algorithm was to firstly train word vectors, instantiated using GloVe, in the context of the data warehouse. Separate word vectors for each of the dimension hierarchies were trained (location and item). These word vectors were then used to generate average word vectors for each unique value in each of the location and item columns, called the column vectors. These artifacts are then persisted and used when a user generates an utterance. The average vector of the utterance is then calculated in terms of the location word vectors and the item word vectors, called the utterance vectors. There are thus three types of vectors: firstly, the word vectors which are trained on the data in the data warehouse, instantiated by GloVe. Secondly, there are the average column vectors, calculated using the average of the word vectors for the words in the columns. Lastly, the utterance vector is the average of the user's utterance words, calculated using the word vectors. Using cosine similarity, the utterance vectors are then compared to the unique column vectors. The nearest column vectors are returned as matches with the utterance; thus, the algorithm returns a value for the location and a value for the item from the utterance.

To test the speed and accuracy of both models, random utterances were generated from the data. To create the random utterances, a survey on question style was filled in by 70 participants. The results indicated that the embedding algorithm performed better in terms of speed and accuracy, with the embedding algorithm performing ten times faster than the heuristics algorithm. The accuracy for the embedding model was 95.6%

compared to the heuristics model with 70.1%.

6.1 Practical Applications

The practical application for this research is to employ these methods in chatbots. Creating a SQL query from the parsed data, and putting APIs on top of it, would allow a star schema sales data warehouse to have the ability to handle natural language queries.

6.2 Future Research

The embedding model is the one which warrants most future work, due to its superior accuracy and speed. Several improvements can be made to the embedding model in future research.

Firstly, a future model should have the ability to cater for situations where no filtering has to happen on any of the dimensions. For instance, an utterance like "What are the sales in Australia?" is currently not evaluated in this research but, in a future model, this should merely mean that the item dimension and the date dimension are ignored in the creation of the query. One suggestion for future researchers would be to have a minimum cut-off of the cosine similarity; for example, if the highest match has a cosine similarity of 0.2, then opt for no_item_selected rather than an item with a low score.

Then, word order and presence can be used to adjust scores of cosine similarities. One of the errors of this research occurred when item words skewed the results of location names, for example, 'red bicycle company' was chosen above 'black bicycle company' when there was a product with the word 'red' in the utterance. When the word order 'black bicycle company' is found in the utterance, but not 'red bicycle company', the vector for 'red bicycle company' should be downgraded in favour of 'black bicycle company'.

For this research, during training, each epoch proceeded with the same batch in the same order. In a future embedding model, different batches could be used in a different order at each epoch during training.

Lastly, a future version of the research can try to implement a Recursive Neural Network, use syntactical parsing and test the matching of the utterance at each branch of the tree, to find the best match. As seen in Figure 6.1, instead of creating one average vector of the utterance "Sales for mountain bikes in the United States", the vectors for 'mountain' and 'bikes' would be combined according to the trained Recursive Neural Network's weights. That combination would be tested across all column vectors. Then the combined vector would be combined with the vector for 'for', which would be tested, and so forth.

This research creates a stepping stone for developing natural language processing abilities on databases by calculating and comparing average vectors. Results in this research indicate that the proposed techniques do warrant future research to continue on this path to build and improve on, specifically, the embedding algorithm.

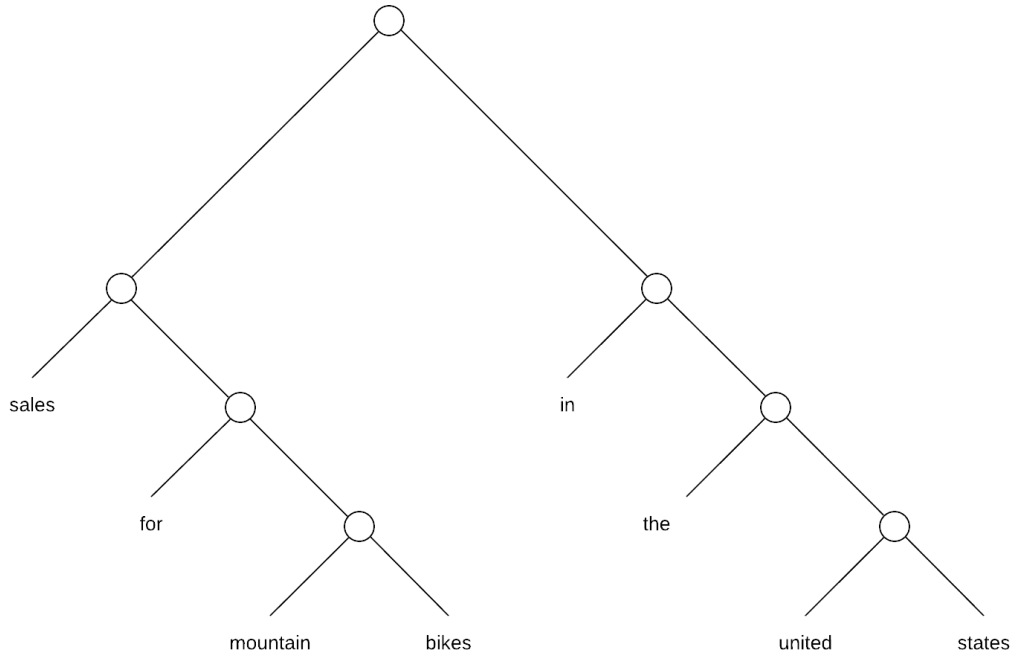

Figure 6.1: Future work can include the use of Recursive Neural Networks to combine vectors.

Bibliography

Amazon (2018). Cloud products. Retrieved from https://aws.amazon.com/products.

Arora, I. and Gupta, A. (2012). Cloud databases: a paradigm shift in databases. *International Journal of Computer Science Issues (IJCSI)*, 9(4):77.

Bird, S. and Loper, E. (2004). Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022.

Bordawekar, R., Bandyopadhyay, B., and Shmueli, O. (2017). Cognitive database: A step towards endowing relational databases with artificial intelligence capabilities. *arXiv preprint arXiv:1712.07199*.

Brad, F., Iacob, R., Hosu, I., and Rebedea, T. (2017). Dataset for a neural natural language interface for databases (nnlidb). *arXiv preprint arXiv:1707.03172*.

Breslin, M. (2004). Data warehousing battle of the giants. *Business Intelligence Journal*, 7:6–20.

Cady, F. (2017). *The Data Science Handbook*. John Wiley and Sons, Hoboken.

Cambria, E. and White, B. (2014). Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, 9(2):48–57.

Charniak, E. (1996). *Statistical language learning*. MIT Press, Cambridge, Massachusetts.

Chomsky, N. (2002). *On nature and language*. Cambridge University Press, Cambridge.

Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, 37(1):51–89.

Danks, J. H. (1981). Language and learning: The debate between jean piaget and noam chomsky (book review). ID: jstor archive 410.2307/1422268.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Donaldson, J. (2018). Comparison of top 6 python nlp libraries. Retrieved from https://www.kdnuggets.com/2018/07/comparison-top-6-python-nlp-libraries.html.

Falle, A. R., Panhalkar, S. K., Jadhav, A. P., Kamble, K. V., Salunkhe, A. A., and Mirajkar, D. V. (2017). Knowledge extraction from database using natural language processing. *International Research Journal of Engineering and Technology (IRJET)*, 04(04).

Gandomi, A. and Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137–144.

Ghahramani, Z. (2001). An introduction to hidden markov models and bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(01):9–42.

Google (2018). Google cloud natural language api. Retrieved from https://cloud.google.com/natural-language/docs/reference/rest.

Grus, J. (2015). *Data science from scratch: first principles with Python*. O'Reilly Media, Inc., Sabastopol.

Han, J., Haihong, E., Le, G., and Du, J. (2011). Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE.

Hu, K., Wu, H., Qi, K., Yu, J., Yang, S., Yu, T., Zheng, J., and Liu, B. (2018). A domain keyword analysis approach extending term frequency-keyword active index with google word2vec model. *Scientometrics*, 114(3):1031–1068.

Hyland, S. L., Karaletsos, T., and Rätsch, G. (2016). Knowledge transfer with medical language embeddings. *arXiv preprint arXiv:1602.03551*.

Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., and Shahabi, C. (2014). Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94.

Johnson, M. J. and Willsky, A. S. (2013). Bayesian nonparametric hidden semi-markov models. *Journal of Machine Learning Research*, 14(Feb):673–701.

Jurafsky, D. and Martin, J. H. (2018). *Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, volume 3. Pearson, London, third edition.

Kaiser, M. (1999). System and method to retrieving information with natural language queries. Retrieved from https://patents.google.com/patent/US6741959?oq=vector+database+natural+language.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436.

Lee, Y.-Y., Ke, H., Huang, H.-H., and Chen, H.-H. (2016). Combining word embedding and lexical database for semantic relatedness measurement. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 73–74. International World Wide Web Conferences Steering Committee.

Li, F. and Jagadish, H. V. (2016). Understanding natural language queries over relational databases. *ACM SIGMOD Record*, 45(1):6–13.

Liddy, E. D. (2001). *Natural language processing*. In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc.

Long, T., Lowe, R., Cheung, J. C. K., and Precup, D. (2016). Leveraging lexical resources for learning entity embeddings in multi-relational data. *arXiv preprint arXiv:1605.05416*.

Loukides, M. (2010). What is data science? the future belongs to the companies and people that turn data into products. *An OReilly Radar Report*.

Lunt, B. M. (2018). A view to the cloud: What really happens when your data is stored on far-off servers in distant data centers. *IEEE Spectrum*, 55(8):40–43.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press, Cambridge, Massachusetts.

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

Mazur, M. (2015). A step by step backpropagation example. Retrieved from https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/,.

McWhorter, J. H. (2004). *The story of human language*. The Teaching Company, Virginia.

Microsoft (2018). Text analytics api. Retrieved from https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Ng, A. (2018). Recurrent neural networks. Retrieved from https://www.coursera.org/lecture/nlp-sequence-models/notation-aJT8i.

Obler, L. K. and Gjerlow, K. (1999). *Language and the Brain*. Cambridge University Press, Cambridge.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Piattelli-Palmarini, M. (1980). *Language and learning: the debate between Jean Piaget and Noam Chomsky*. Harvard University Press, Cambridge, Massachusetts.

Pinker, S. (2007). *The stuff of thought: Language as a window into human nature*. Penguin, New York.

Pinker, S. and Poeppel, D. (2000). Words and rules: the ingredients of language. *Nature*, 403(6768):361.

Rajpurkar, P. (2018). Squad2.0. Retrieved from https://rajpurkar.github.io/SQuAD-explorer/.

Rob, P., Coronel, C., Silberschatz, A., Korth, H., and Sudarshan, S. (2006). Database systems: Design, implementation. *Management.Seventh Edition.Course Technology*.

Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph databases*. O'Reilly Media, Inc., Sebastopol.

Russel, S. J. and Norvig, P. (2010). *Artificial intelligence: A modern Approach*. Pearson Education Limited, Malaysia.

Shaikh, A., Phalke, G., Patil, P., Bhosale, S., and Raghatwan, J. (2016). A survey on chatbot conversational systems. *International Journal of Engineering Science*, 3117.

Silge, J. and Robinson, D. (2016). tidytext: Text mining and analysis using tidy data principles in r. *The Journal of Open Source Software*, 1(3):37.

Socher, R. (2018). Tree recursive neural networks and constituency parsing. Retrieved from http://web.stanford.edu/class/cs224n/lectures/lecture14.pdf.

Swart, H. D. (1998). *Introduction to natural language semantics*. Cambridge University Press, Cambridge.

Turing, A. M. (1950). Mind. *Mind*, 59(236):433–460.

Vasudevan, V. and Wicke, M. (2018). Word2vec github repository. Retrieved from https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/word2vec /word2vec_basic.py.

Wieting, J., Bansal, M., Gimpel, K., and Livescu, K. (2015). Towards universal paraphrastic sentence embeddings. *arXiv preprint arXiv:1511.08198*.

Zeroual, I. and Lakhouaja, A. (2018). Data science in light of natural language processing: An overview. *Procedia Computer Science*, 127:82–91.

Zhang, L., Wang, S., and Liu, B. (2018). Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1253.

Zhong, V., Xiong, C., and Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Zucchini, W., MacDonald, I. L., and Langrock, R. (2016). *Hidden Markov models for time series: an introduction using R*. CRC Press, Boca Raton.