Universität Zürich[UZH]

Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Arts
der Philosophischen Fakultät der Universität Zürich

Information Retrieval Chatbots

Verfasserin/Verfasser: Christof Bless
Matrikel-Nr: 14-710-982

Referentin/Referent: Christof Bless

Betreuerin/Betreuer: Dr. Tanja Samardzic

Institut für Computerlinguistik

Abgabedatum: 30.06.2018

Abstract

Conversational bots are a big topic in fields like customer care or artificial personal assistance. Chatbots give us new ways to access all sorts of information quicker and more reliable. Querying data trough natural language is becoming reality and Information Retrieval Chatbots are making it possible.
This thesis gives a short overview on the general topic of chatbots and discusses different architectures for Information Retrieval Chatbots. Machine learning and rule-based approaches will be introduced and compared. In the end I will present an attempt to build a German question-answering chatbot and evaluate it on a test set of questions.

Zusammenfassung

Konversationsbots sind ein wichtiges Thema für Anwendungsgebiete wie Kundenbetreuung oder künstliche persönliche Assistenten. Chatbots eröffnen uns neue Wege, um verschiedene Arten von Informationen schneller und verlässlicher zu erhalten. Daten mit natürlicher Sprache abzufragen wird zur Realität und Information Retrieval Chatbots werden entwickelt, um dies zu ermöglichen.
Diese Bachelorarbeit gibt einen kurzen Überblick über das allgemeine Thema der Chatbots und erörtert verschiedene Architekturen von Information Retrieval Chatbots. Machine learning und regelbasierte Vorgehen werden vorgestellt und verglichen. Danach werde ich einen Versuch präsentieren, einen deutschen Chatbot zu erstellen und evaluiere ihn auf einem Test-set von Beispielfragen.

Acknowledgement

First and foremost I want to thank my supervisor Dr. Tanja Samardzic who mentored me not only during this thesis but already in so many other occasions which has always been a fantastic experience. It has been thanks to her constant support and encouragement that I managed to stick to my desired project and pull it through. I am very thankful for all her precious advice.

Dr. Samardzic has also been my connection to Claudiu Musat and Ignacio Aguado at Swisscom who greatly helped me develop my practical experiments. I want to thank them for their time and patience in the meetings we had during the 5 months. It was very helpful to rely on their phenomenal work. I immensely profited from their advice and all the clever tips.

Contents

List of Figures

List of Tables

List of Acronyms

AIML	Artificial Intelligence Markup Language
A.L.I.C.E.	Artificial Linguistic Internet Computer Entity
API	Application Programming Interface
CBOW	Continuous Bag of Words
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network
SQL	Structured Query Language
XML	Extensible Markup Language

1 Introduction

1.1 Motivation

> It is said that to explain is to explain away. This maxim is nowhere so well fulfilled as in the area of computer programming, especially in what is called heuristic programming and artificial intelligence. For in those realms machines are made to behave in wondrous ways, often sufficient to dazzle even the most experienced observer. But once a particular program is unmasked, once its inner workings are explained in language sufficiently plain to induce understanding, its magic crumbles away; it stands revealed as a mere collection of procedures, each quite comprehensible.

This quote by Joseph Weizenbaum, developer of ELIZA, the first convincing conversational chatbot, describes a process that many people working with computers know. A program or system, impressive and intimidating at first sight, becomes easy to understand once we start dissecting it into the little pieces that make it up. We start to see how the different parts interact and form the program we are seeing. In the end all that is left to say is: "I could have done that." That's what the motto of this thesis shall be. With my thesis I want to explain how chatbots work and uncover their magic.

Talking is one of the best indicators of intelligence to us humans. Language evolved as the arguably most elaborate result of genetic evolution in our species. No other species has mastered this ability which has put us in a privileged position. But it remains a very mysterious privilege. What is it that makes us able to think and speak in language? The origin of language and intelligence is obscure. Since their beginnings computers are thought to give answers to these questions.
For me chatbots are the closest thing to general artificial intelligence we have right now. Chatbots emulate a human conversation partner. I think once a program can get perfect at emulating human language behaviour, it will start to become clearer to us what it means to be human. Also what intelligence means and if it is neces-

sary to be human to be intelligent. It is exciting to think that one day, it might be possible to talk to a being that doesn't exist in the real world.
When I started programming 4 years ago, developing a chatbot was one of my first dream projects. I had several small attempts but never really got through with the plan, because it seemed like an overwhelmingly big project. Now for my bachelor thesis I will take another go. I will have a look at the topic of chatbots as a whole and then specialize in a practical attempt trying to train a learning German Information Retrieval Chatbot.

1.2 Research Questions

The key questions of my thesis are related to the practical work around it, which was developed together with Swisscom. The company is putting a lot of research into developing chatbot technology and they agreed to support me with some of their knowledge. I used part of their chatbot system, and changed some of it, to come up with my own version of a German chatbot.
During this process several path choices had to be taken and questions came up. These are the most important questions that I had to answer to arrive at my end-goal of the German chatbot:

1. Which chatbot architecture is better for information retrieval, rule-based or statistical machine learning?

2. What does it take to train a German chatbot that performs similar as an English one on the same dataset?

3. Is there an advantage using a neural network architecture for our bot?

As we will see in the next chapters, the first question is a relevant one, since machine learning is not yet the most reliable choice when it comes to general architecture of typical conversational bots. Information Retrieval bots on the other hand, have a few properties which make them more interesting for the use of machine learning. In the second chapter I will mostly answer this question by showing the advantages and disadvantages of rule-based and machine learning approaches in chatbot development.
The second and third question will be addressed in the third chapter. I will explain some of the decisions we took in the design process of the German question-answering bot. This will shed light on the question what it takes to train such a system and why neural networks perform good at it. I will discuss why I think that the choices were

the best ones for our scenario and evaluate if the outcome was positive compared to the results of the English system.

1.3 Thesis Structure

In this thesis I will review the state of the art in chatbot technology and report my own experiences with developing a German Information Retrieval Chatbot. The first chapter serves as an introduction to my motives and my goals with this thesis. The research questions are defined which will be answered at the end of the document. In the second chapter I will give a general introduction as to what a chatbot actually is. I will follow up with an extensive overview of the history of such programs. Beginning with the early conversational technologies until the situation of today. The landscape of modern chatbot applications is changing very rapidly in the last years and still things are shifting very much. I will give a short overview on the different types of chatbots that we can find in industry and academia.
Different kinds of chatbot architectures will come up and I will give detailed examples with explanations of the most important ones.
Chapter 3 introduces my practical work and the collaboration with Swisscom. I'll explain the setting that I found at Swisscom and the background on which the project originated. I'll give an overview on the methods that were used and the different stages of the design process. In the end I will present my evaluation results and discuss them.
In Chapter 4 I give my conclusion and answer my research questions.

2 Chatbots

2.1 What is a chatbot?

A chatbot is a conversational agent that interacts with users through natural language. As the name 'bot' suggests, the agent is a robot or more specifically a computer program. It receives input from a conversational partner in the form of a question or other natural language expressions. It outputs an answer or other expressions to uphold a conversation.
It depends on the type of chatbot whether the focus of the conversation is to answer a question or just to keep some smalltalk going. A chatbot can also be a personal assistant and perform certain actions. Chatbots can be guides that help achieving a task. There are a lot of different use cases for chatbots and all of them have different requirements to architecture and capabilities of the bot.
Microsoft CEO Satya Nadella said in a keynote given at a Microsoft developer conference in March 2016: "Bots are the new apps".[1] She is referring to the way companies are using bots to distribute their services or information via dedicated bots instead of apps. When Mark Zuckerberg announced Facebook Messenger would open up to branded chatbots at the F8 conference in April 2016, "1-800 Flowers" was the example he used in his demo, a bot that lets users order flowers just by chatting.[2] According to Ted Livingston, CEO and founder of the messaging app Kik, "People think bots are about chatting. They're just a better way to deliver software. It's just a user interface."[3]
More generally spoken bots can simplify the process of receiving any kind of service, which was only possible by installing a dedicated app beforehand. A chatbot can respond directly to a user's needs in the familiar environment of chat applications. They can be a natural language interface with many kinds of software, which eliminates the need to adapt to new interfaces or procedures for the user.

[1] https://eu.usatoday.com/story/tech/news/2016/03/30/microsof-ceo-nadella-bots-new-apps/82431672/ - 22.06.2018

[2] http://www.businessinsider.com/live-facebooks-f8-conference-2016-4

[3] https://econsultancy.com/blog/68532-the-case-for-chatbots-being-the-new-apps-notes-from-websummit2016/ - 22.06.2018

2.2 History

The idea of a chatbot is about as old as computers are. In 1950 Alan Turing proposed the Turing test to assess the ability of a computer to act like a human.[Turing, 1950] He envisioned artificial intelligence as machines which can take the role of conversational partners. He thought of computers giving creative, complex responses to human input instead of just trivial preprogrammed reactions. It should be remarked that Turing formulated these thoughts only one year after the first computer program had even been run on an actual machine. And these programs were only used to compute numbers and solutions of equations. The desire to talk to computers which were initially designed to perform arithmetic operations, was conceived very early in the history of computers.

Turing Test The idea of the Turing test was published in the paper 'Computing Machinery and Intelligence' by Alan Turing. [Turing, 1950] He wanted to create a method to prove human-like intelligence in a machine.
The test involves two human beings and one machine. One of the two people acts as an interrogator while the other acts as a chat-partner. The machine then acts as another chat partner. The setup involves two screens with keyboards where questions or statements can be typed to start a conversation. It is hidden to the interrogator which of the screens has a human being behind and which a machine.
After conversation, the interrogator is asked to decide which of the chat partners was human and which wasn't. The Turing test is passed if no statistical preference can be found towards the 'real' chat partner.

Talking computers weren't an absurd concept at the time. Many early thinkers in the field of computer science were already reflecting on intelligent machines. John Von Neumann, who invented the basic architecture of all our computing devices today, wrote a book where he compared computers and brains. [Neumann, 1958] He, like many others, was intrigued by the question whether or not computers could think creatively and how such intelligence could be proven.
The quest for artificial intelligence was one of the main drives for early developments in chatbot technology. It all started with simple programs that were based on keyword matching. They would search for certain keywords in the users input and then return preconfigured answers based on that. It was especially before the arrival of graphical interfaces that many developers would create such chatbots mainly for fun. [Shawar and Atwell, 2007]

ELIZA One of the first notable efforts in the field was the creation of Joseph Weizenbaum's ELIZA [Weizenbaum, 1966], a computer program that was able to hold a conversation pretending to be a psychotherapist. In the program, rules are used to transfer an input question into a response. Weizenbaum defined decomposition and reassembly rules which could be activated by certain keywords.
One of the noticeable features of the program was that it often just rephrased the question into a counterquestion. Very rarely a concrete answer to a question was given. The conversation with ELIZA was usually set up in the context of talking to a Rogerian psychotherapist, which is a form of psychotherapy, where the therapist just guides the patient through their self-reflection to finally overcome their problems through acceptance. In this setting the chatbot's tactic of avoiding direct answers did not take away from the illusion that ELIZA was indeed human.
One example for a conversation could start with the input sentence "It seems that you hate me". ELIZA matches the keywords "you" and "me" in this sentence. These are connected with the decompositon rule "(0 YOU 0 ME)". 0 stands for an infinite number of words like the wildcard symbol in regular expressions.
The reassembly rule goes like this: (WHAT MAKES YOU THINK I 3 YOU). The 3 in this stands for the third component of the decomposition rule. All the words that stand between ME and YOU.
The bot then answers: "What makes you think I hate you?" In a psychotherapy context such conversational turns looked very convincing. Many people actually believed that the computer understood their problems.
In later years many chatbots followed up on similar ideas where the context contributed immensely to the illusion of humanness of the program. One notable example of this was Parry, a bot simulating a paranoid mental patient. [Shawar and Atwell, 2007]

Loebner Prize In 1990 Hugh Loebner established a competition for a successful implementation of the Turing Test. $100'000 were initially offered as a prize for the first computer that can not be distinguished from a human being in the ability of upholding a conversation. Later the requirements also included being able to interpret textual as well as visual and auditory input. The first bot that achieves the goal just on text without visual and auditory systems can claim a silver medal and $25'000. A bronze medal and an annual prize of $2'000'-$3000 are given every year for the computer which seems to be more human in relation to the other competitors, regardless of how good it is absolutely. It is the first known competition that represents a formal instantiation of the Turing test. [Mladenić and Bradeško, 2012]

The competition is running since 1991 annually with slight changes made to the original conditions over the years. The goal of it is to design a chatbot that has the ability to drive a conversation. During the chat session, the interrogator tries to guess whether they are talking to a program or a human. The setting is an implementation of the above-mentioned Turing test. The interrogator is presented with two screens on which he's presented with interactive conversations. One having one of the chatbots behind, the other a voluntary person, who tries to answer questions as naturally as possible. After a ten minute conversation the interrogator has to judge which one was the human person. Humanness of the bot must be evaluated on a scale from 1 to 4. According to this judgement, the more human chatbot is the winner. No chatbot has ever passed the test to win the Loebner Prize gold medal. Some chatbots have scored a maximum of 3 out of the 12 judges believing they were human.
The first winner of the bronze medal in 1991 was *PC therapist* by Joseph Weintraub, which was largely based on ELIZA. He won the competition 4 times with this bot.

A.L.I.C.E. With improvements in machine learning techniques and availability of chat corpora, better chatbot architectures arose. Most notably the ALICE/AIML architecture. A.L.I.C.E. or Alicebot was the name of the bot that won the Loebner Prize in 2000, 2001 and 2004. It was developed by Richard Wallace. He refined Alicebot's code over the years and made it open-source. With this came the public release of the AIML standard. (see section 2.3.1) AIML stands for Artificial Intelligence Markup Language and it is an XML standard to write ELIZA-like conversion rules to transform chatbot questions into answers.
The development of AIML led to many open-source frameworks that allowed the fast and simple collection of AIML rules. Many of these frameworks make setting up customizable bots very easy. The character and domain of a bot can be formed through its conversation rules. Hundreds of Alicebots clones have been created in the years after. Many tools became available that abstract the creation of a chatbot so much that also people without programming experience can develop simple chatbots.

Cleverbot As of today, most of the competitors in the Loebner prize have been using an AIML architectures. One of the exceptions being 2005 and 2006 prize winner Jabberwacky, which later evolved to the popular chatbot *Cleverbot*. According to its inventor Rollo Carpenter, Jabberwacky learns how to answer from a big store of learned answers and uses pattern matching techniques to find the appropriate ones. On the website it is written:

> Jabberwacky stores everything everyone has ever said, and finds the most appropriate thing to say using contextual pattern matching techniques. In speaking to you it uses only learnt material. With no hard-coded rules, it relies entirely on the principles of feedback. This is very different to the majority of chatbots, which are rule-bound and finite.

Chatscript In recent years, another standard for the definition of chatbot conversion rules was introduced. *Chatscript*. Chatscript allows more compact rule definition than AIML. It was brought up by Bruce Wilcox who won the Loebner prize three times with bots based on Chatscript architecture. I will give an example of Chatscript syntax in section 2.3.1.

Mitsuku Mitsuku, created by Steve Worswick, is considered one of the best conversational bots in existence. It won the Loebner Prize in 2013, 2016, and 2017. Mitsuku is based on AIML architecture. In addition to that it has some basic reasoning skills. The bot has been worked on since 2005 and updates to the AIML rules are scripted based on conversation logs with users.
In 2015, Steve Worswick announced that his bot had 7 million interactions per month on the Kik messenger platform alone.[4] Mitsuku has been launched on Pandorabots.com which is a chatbot hosting platform that lets people easily create, develop and deploy AIML chatbots. Pandorabots allows direct integration of chatbots into many messengers services like Whatsapp, Facebook Messenger or Kik.

As stated in the introduction, chatbots can be easily integrated into the Facebook messenger app since 2016. As of July 2016, there were more than 11'000 bots on the Facebook Messenger platform.[5]
Chatbots are used by many industries today, most prominently in the field of E-commerce. Other fields where Chatbots are being used are: Media & Entertainment, Customer Service, Traveling, Games, Financial Information, Personal Assistants and Food & Restaurants.

In the last 2-3 years interest in chatbots has skyrocketed. User Trends are showing that People install less and less different apps. Messenger apps are by far the most installed apps. And people are not opposed to centralize a variety of functions on their messenger apps. One of the best examples for this trend might be the massively

[4] https://twitter.com/MitsukuChatbot/status/671095095287029761 - 28.06.2018

[5] https://econsultancy.com/blog/68532-the-case-for-chatbots-being-the-new-apps-notes-from-websummit2016/ - 28.06.1018

popular chinese messaging app WeChat which allows users to find social networks, gaming, mobile payment and many more features all inside a messenger app. The idea to use chatbots to give people a means of direct interaction with a company has gained a lot of traction in the latest years. Nearly all of the big messaging apps allow easy integration of custom bot programs into their system.

2.3 Types of chatbots

With the variety of use cases of today's chatbots there come many differences in purpose and structure of these bots. If we are talking about chatbots there are different categories which we might want to distinguish. First let's consider the purpose of the chatbot. The goal of a conversational bot might be to entertain a user and excel at smalltalk. Other chatbots are intended to inform their chat partners about a specific topic. Then there are chatbots that answer questions and chatbots that follow orders.
The structure can also have many different forms. We have already heard about conversational chatbots and their mostly rule-based backgrounds. Besides that there are, of course, bots that involve different degrees of machine learning. This starts at things like intent classification and goes all the way to strict sequence to sequence neural network architectures, which generate whole responses just from a question.
In terms of categories, these are some classification bullet points:

- conversational vs. information retrieval
- long vs. short conversation
- closed domain vs. open domain
- rule-based vs. machine-learning

On the one hand there are the conversational bots like the ones competing for the Loebner prize. They are designed for long conversation in open domain and often their answers are witty but very shallow.
In the industry, information retrieval chatbots are in demand. Many companies store their knowledge in relational databases. Whether it is user-information, statistics, financial information or general data. An information retrieval chatbot is a specific type of bot which lets a user interact with such a data store through natural language conversation.
These specific chatting bots have found their way into daily life in recent years. In most cases it is more a question-answering system than real conversation and

interactions are typically very short. The conversations have a certain goal and meaningful answers are key.
Companies like Lufthansa, H&M or Zalando are currently using such chatbots to answer simple questions of their users. Easy and obvious answers can be preconfigured by a developer. Questions like 'Where do I find the men's underwear on your website?'
The domain can be very closed but it is important that the bot understands all sorts of questions. Conversational tricks are not a good option to hide the fact that a system does not understand a question correctly. For all the possible questions that are generally answerable by our data we want to connect the question with the correct answer. This is often achieved with some generalizing machine learning models.
Let's consider this more complex scenario: An information retrieval chatbot will have to retrieve data from a database. For example someone might want to know: "On which weekdays do flights to Athens depart from Frankfurt?". It is practically impossible to preconfigure answers to questions like this especially since the information can change all the time. The chatbot has to rely on some sort of knowledge graph or data storage that it can query for this kind of information.
The bot transfers the input sentence into a database language representation, looks up the query and responds with the result. Such chatbots serve as a mere interface between the user and the data where the user can use his own words to retrieve information without any other abstraction.
In the following we will have an in depth look at some of the techniques used in rule-based chatbots as well as machine learning approaches.

2.3.1 Rule based approach

Starting with Joseph Weizenbaums ELIZA, the majority of chatbots created for real world usage have been developed around a set of rules. Every winner of the Loebner Prize competition up to date has used a rule based approach paired with pattern matching on the input sentence to produce output sentences. Meanwhile machine learning approaches to chatbot architecture are not underdeveloped at all. The reason why rule-based approaches are dominating the Loebner competition is that with rules it is much easier to build a chatbot that pretends to be human. Actually it is in a way human because it just outputs sentences that were defined by human programmers.
Machine learning bots on the other hand are often easy to unmask because they make mistakes in sentence structure and logic. It is however very questionable if a

deterministic chatbot defined by a finite set of rules can ever pass the Turing Test. There are just too many imaginable possibilities of conversational turns in order to cover them all with rules.
Since rule based chatbots make up the majority of bot-systems today, we will have a look at the two major frameworks that are used to build these bots. AIML and Chatscript. AIML is a dialect of XML. It was one of the first bot scripting languages and is still widely used today. AIML stores all the rules that are put into a chatbot in AIML objects. It was originally developed by Richard Wallace who released his famous chatbot A.L.I.C.E. under GNU license. (see section 2.2) Community work on the open source code base of A.L.I.C.E. led to the creation of AIML 1.0 in 2002 and AIML 2.0 in 2013 which has some more powerful features. The purpose of the AIML language is to simplify the job of conversational modeling. The scripting boils down to the most basic idea of building a chatbot: Defining a set of input and output pairs.
A very simple AIML Rule can look like this:

```
<category>
<pattern>WHAT IS YOUR NAME</pattern>
<template>My name is Michael N.S Evanious.</template>
</category>
```

Individual Rules are marked by 'category' tags. The 'pattern' tag matches an input sentence. It also features wildcard symbols like * and _. The *template* tag defines the answer the bots gives. It can be just plain text, but it is also possible to define variables that change depending on input and context. Inside *category* tags we can also find *that* tags and *topic* tags which can define a certain context in which the rule should be used.

Chatscript is a more modern chatbot scripting framework. It allows more compact rule definitions and powerful features like extensive use of ontologies, support of relational databases like PostgreSQL and the ability to read JSON structures directly taken from the web.
An example of a Chatscript script file would be the following:

```
topic: ~pets (dog cat pet animal bird fish snake)
?: ( << you like snake >> )
 I love pythons except ^"Python" (the programming language)
t: Do you have any pets?
 a: ( ~yes ) Great. You like animals.
 a: ( ~no ) You don't like animals?
```

```
a: ( cat ) I prefer dogs
a: ( [parrot bird canary finch swallow] ) Birds are nice.
```

In Chatscript, we define files for all the topics our bot should cover. Lines starting with 't:' define things the bot says without being asked. At the core rules are defined very similar to AIML. We have these major types of rules:

- s: means the rule reacts to statements.
- ?: means the rule reacts to questions.
- u: means the rule reacts to the union of both.

The pattern which activates the rule is given in brackets and the answer comes after that. a: to q: are called rejoinders. They can only be activated if the rule before them has been activated. So only after the question 'Do you have any pets?' is formulated, the bot will answer to any input involving 'yes' with 'Great. You like animals.'
In addition to that there are many sorts of variables and concepts one can define. It is generally a bit more complex than AIML but comparing a similar chatbot in any of the languages, Chatscript uses fewer distinct rules.

With all these rule-based systems it is tempting to think that it is an immense work to put up enough rules to be prepared for practically anything a person could say. But in reality the prize winning bots only use a few thousand rules and are already very convincing. This stems from the fact that these rules already allow a certain degree of generalization with the definition of topics and wildcards. The number of concepts and topics people generally converse about is surprisingly small. Especially if we compare it with the number of syntactically correct sentences that could be formed combinatorially with our vocabulary.

2.3.2 Machine learning approach

The problem with pattern based heuristics in chatbot development is that the patterns have to be manually crafted by a developer. Even though these systems can be very convincing and human-like they can quickly be overwhelmed by unforeseen input.
Especially if a bot has a certain function and does not just serve as an entertaining conversational bot. If the bot should provide some information to the user, we can't just rely on some cheap tricks to hide the fact that the computer in fact does not

understand what we are talking about. It is necessary that the bot recognizes what the user wants. One approach to achieve this is intent classification.
If we imagine a customer service bot at some company for example, we might want to ask it, if refunds are possible. There are a lot of different ways we can express this question in language. “I want a refund!”, “Can I request a refund if I don’t like the service?” or “What is your refund policy?”. All these expressions share the same intent. To predict this intent based on the input an intent classifier can be used. Given a list of intents or categories and pairs of example question and associated intent, the classifier can learn to associate the correct intent to questions.
At the moment many of the big cloud providers offer customizable intent classifiers. Facebook has wit.ai, Google api.ai, Microsoft LUIS and on Amazon it is Lex. They use similar strategies in high level scripting like AIML or Chatscript. The difference is that you can define a few input sentences and the underlying intent which will also determine the output. The system will then generalize any given sentence to one of the possible intents you created, just by using supervised learning algorithms on your examples.
Another learning method is using deep neural networks on huge conversation logs to train a sequence to sequence model. Given an input sentence, the system will then be capable of predicting a whole output sentence or answer using just an encoder-decoder model.[Pereira and Luisa]

3 A German Chatbot

3.1 Definition of Task

I have introduced the concept of an information retrieval chatbot in section 2.2. In particular the type of chatbot that uses machine learning techniques to convert natural language into database query representations.
At Swisscom, Switzerland's biggest telecommunications provider, this kind of chatbot is a topic of ongoing research. It is especially areas related to customer care where it is very imaginable that chatbot technology could increase quality and efficiency of services. The company needs to assure constant support for their customers in cases of technical issues, consultation and all sorts of question-answering. Depending on the time of the day and the general situation it is necessary to wait for a long period of time to speak to a customer care employee. This is one of the biggest problems when it comes to customer satisfaction.[1] Integrating chatbots into the process can presumably make it easier for all the parties involved.
Additional to that, we have the problem of language. While there is an abundance of chatbots in English, other languages have only seen little development in this area. Considering that Switzerland has four official languages, multilingual chatbots would make a lot of sense at Swisscom. People feel a lot more comfortable and satisfied when they can communicate in their native language.
Many short and easy questions can potentially be answered thoroughly and correctly by relatively simple chatbots. This gives employees more time to focus on harder cases and provides customers with a fast and uncomplicated response. Customers do not have to be on the phone for several minutes to get a simple answer.
As explained earlier, a chatbot can also handle slightly more complicated cases, where the correct answer can only be found in a database. It is possible to train a machine learning model that generates SQL expressions given a natural language question and look up information in a big table. A well trained information retrieval bot could potentially be added to the current question-answering setup in customer

[1] https://www.swisscom.ch/de/business/enterprise/themen/digital-business/des-2017-008-servicesmit-kuenstlicher-intelligenz.html - 30.06.2018

service at Swisscom.

To get hands-on experience with the actual research that is being conducted in this

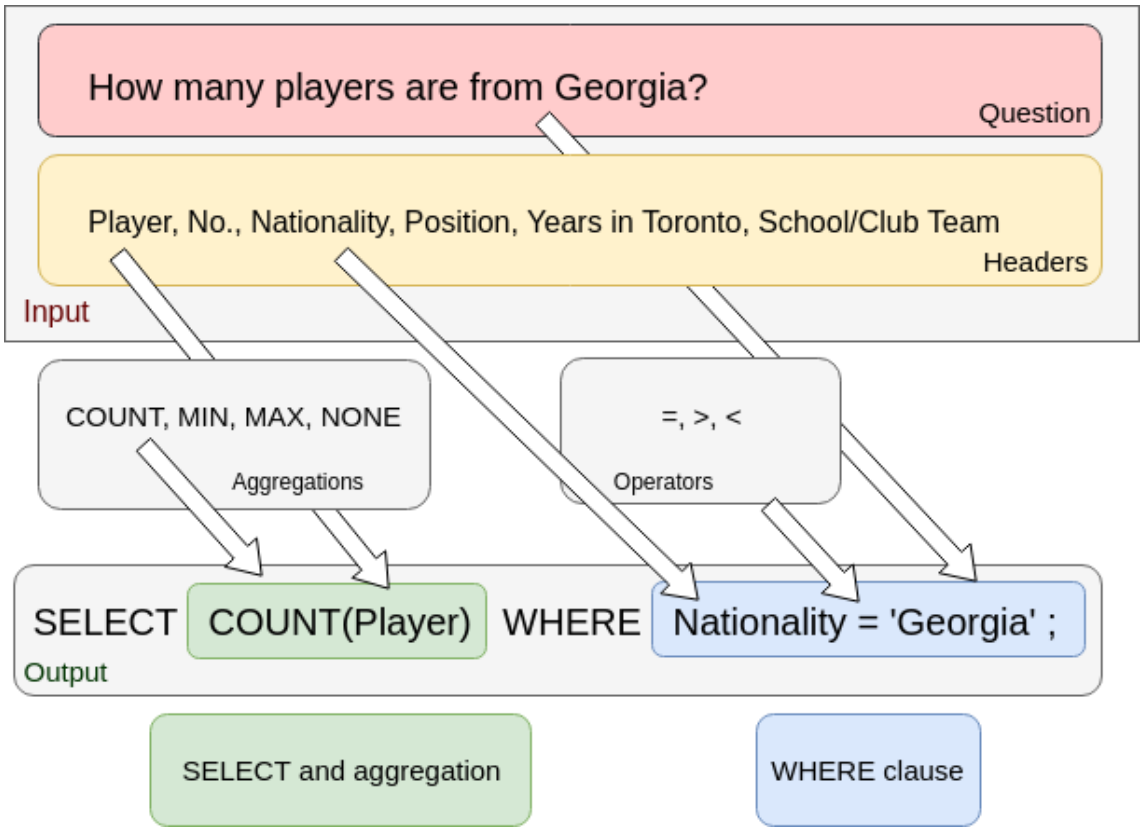

Figure 1: Overview of the SQL generation task

area we arranged a short collaboration with Swisscom's AI group. I was introduced to Claudiu Musat and Ignacio Aguado and their ongoing chatbot system experiments. Their experiments involve a neural network chatbot architecture which is able to retrieve answers from a database of tables extracted from Wikipedia. The project is a proof of concept for the task of natural language to SQL conversion. Their experiments involve a bot, that can be provided with a question and predicts an SQL expression through several neural networks which have different properties based on the specific function that they serve in generating the SQL query.

Consider this example: The question "How many players are from Georgia?" is translated to the SQL query "SELECT COUNT(Player) WHERE Nationality='Georgia';". The column name 'Player' and the WHERE clause "nationality='Georgia' " have to be predicted from the inputs. The aggregation function COUNT is chosen from the set of possible aggregation functions in SQL syntax. (See Figure 1)

These elements are generated by a series of neural networks. Some of the networks just predict one element out of a series, others encode a series of words into a vector and still others decode or generate a series of words from an encoder output. In the

end, outputs of several networks form the full SQL query string. In the following sections, all of these individual parts will be taken apart and explained.
Our goal was to test the system's ability to adapt to new languages. What we had in mind here, was to conceptualize the idea of multilinguality, which would be a very helpful feature of any chatbot as mentioned above. In this case I worked with the biggest language in Switzerland - German - trying to find out where the difficulties might lie and find possible solutions.
In particular we would come up with a way to generate the equivalent training material for the bot in German as it had in English and see if evaluation results can be at least comparable to those of the English system. The limiting factor here is the quality of the training corpus we begin with and especially the quality of the translations that are generated. As I will explain in the next section, the corpus we are working with is created quite artificially and therefore not a very good example of natural language. Therefore in the act of translating this corpus the connection between the question sentences and the aligned SQL expressions will get lost to a certain degree.
In German, sentences are generally longer than in English and things like syntactical variation or compound nouns could make the task of learning the corresponding SQL query for a German sentence harder than in the English case.
The question that needs to be answered is, whether automatically translating the training corpus gives useful results in constructing a chatbot system in a different language.

3.2 Groundwork

In my practical evaluations I have not been setting up anything from scratch. I largely relied on previously built neural network training architecture. Swisscom largely follows the approach of Zhong et al. [2017] and also Xu et al. [2017] to train and evaluate their early experiments. In the following I will describe some of the groundwork underlying my experiments and the data being used.
Zhong et al. [2017] provide the WikiSQL data set which is used to develop the chatbot. Zhong et al. [2017] also describe a neural network architecture to translate natural language sequences to SQL which is the basis of my practical work. I will explain the content of the Seq2SQL project in more detail later. In general they used a big corpus of natural language question and SQL expression pairs to train their system to generate SQL expressions from natural language questions.
The idea of the presented practical work is to adapt the corpus and system to

German language. This means translating the natural language questions to German with a machine translation service, then preprocessing the sentences and retraining the Seq2SQL system with some adaptations. I train a number of different models with different training data and system parameters. Then the models are all used to predict SQL expressions in a given test set of sentences.
Subsequent evaluation of the results will show what setting is best for the task and if the resulting bot can compete with the original English one.

3.2.1 Seq2SQL

Seq2SQL is a project described in two research papers published by Salesforce research team in 2017.[Zhong et al., 2017; Xu et al., 2017] The task they were trying to solve is the generation of SQL queries from natural language expressions.

WikiSQL The WikiSQL data set is a data set that comprises 80'654 manually produced examples of natural language questions aligned with SQL queries generated from 24'241 tables taken from Wikipedia. It was gathered using the crowd sourcing platform Amazon Mechanical Turk where workers were asked to form English sentences from SQL paraphrases.
To start with, a subset of the Wikipedia tables is used to generate the SQL queries. So for example a table about a certain basketball team might give us many possible queries like this one:

```
SELECT COUNT(Player) WHERE Nationality='Georgia';
```

Listing 3.1: SQL with aggregation

This is basic SQL syntax to say that we want the content of column "Player" in the rows where column "Nationality" has the value "Georgia" and then to count together all the individual cells that follow these constraints. The query counts all the players that come from Georgia.
COUNT is an example of an aggregation function. Aggregations are cases where an aggregate has to be computed from a number of values. In the data we can find four types of aggregations: COUNT, MIN, MAX or None (no aggregation). No other functions were used in the SQL expressions that were generated for creating the WikiSQL corpus. This is quite arbitrary and in a real scenario there would be more aggregation functions to learn, to get all answers from the data.
To extract the SQL strings, they only chose tables big enough and well formatted.

In a next step an English language paraphrase of the SQL expression is automatically generated using a template. The SQL expression from above for example could be translated to "How many player when the nationality is Georgia?" which is easily derivable from the SQL string. After that the freelance Amazon worker, called "turker" comes in. He forms the prepared question into a proper English sentence. The turkers also control whether the result of the SQL query is actually a valid answer to the natural language question.

In Zhong et al. [2017] a neural network architecture is presented to translate from the natural language question to the SQL expression. I will explain this architecture in detail further down. In the earlier paper, Zhong et al. [2017] use policy-based reinforcement learning in the form of immediate query execution on the database. In the next paper Xu et al. [2017] describe a pointer network without reinforcement learning which leads to a performance increase. For the experiments at Swisscom, structures from the earlier approach are used. For specific parts of the network architecture like the WHERE clause prediction, they did not use reinforcement learning but a pointer network as described in Xu et al. [2017].
The general Seq2SQL architecture is built around a deep neural network model called the pointer network. [Vinyals et al., 2015] It consists of an encoder two-layer bidirectional Long Short-Term Memory network and a decoder two layer unidirectional LSTM network. Pointer networks are implementations of Recurrent Neural Networks (RNN). They are used to solve sequence-to-sequence problems. A pointer network uses the mechanism of neural attention as a pointer to select a member of the input sequence as the output.
There will be an in-depth explanation of the different pointer networks that are used to generate the output in section 3.3.2. In our case, input to the pointer network is provided in form of all the tokens of the question and all the tokens of possible column headers in the corresponding table. More specifically word embeddings of tokens are used as an input layer. (For more information about embeddings, see section 3.3.3)
All the embeddings of the input set are encoded by a neural network so that we get an encoder output for each of the words in the input sequence. On the encoded input sequence a decoder network is built. At each decoder step the decoder LSTM network takes as input the query token generated during the previous decoding step and outputs a state. With this state a scalar attention score for each one of the input tokens is computed, which determines the output token.
In a real world application, we assume that we know beforehand which table contains the information that the user is seeking or that all knowledge is stored in the same table. Consequently the table name (FROM statement) is not part of the predicted

output. This also means that we should know all the possible column names that are part of the input to the network.
Let's consider another basketball example:

```
SELECT result WHERE place='Los Angeles' AND date='05.09.1993';
```

Listing 3.2: Example SQL expression

Given an input sentence like "What was the result of the game in Los Angeles on 05.09.1993?" and the column names "place", "result", "date", "number", etc., we want to generate the above expression. Due to the simple and consistent syntax of SQL, the hardest things that need to be predicted here are actually the slots with column names and values. The SQL specific tokens like "SELECT", "WHERE" etc. are pretty much given with the exception of aggregation functions like COUNT or MAX. So what has to be predicted are actually the SELECT column, possibly an aggregation on the selection and the WHERE clause.
One problem that arises with SQL is that different queries can often have equal results. Let's look at this query for example:

```
SELECT place WHERE date='05.09.1993' AND place='Los Angeles';
```

Listing 3.3: Example SQL expression equivalent to 3.2

This SQL expression will yield exactly the same result as the one above, even though the ordering of the constraints is switched up. They are semantically equivalent. However in automatic evaluation they won't be seen as equivalent. This phenomenon can affect the performance of the model significantly.
To simplify the task and prevent this problem Xu et al. [2017] introduce a canonical representation of SQL strings. The above query for example would be formalized to the following construct:

```
{
 "aggregation":None
 "select": "results",
 "conditions": [
    ["date", "=", "05.09.1993"],
    ["score","=",50]
 ]
}
```

Listing 3.4: Canonical representation of SQL query

The neural network now only fills this data structure. The SELECT column as well as the column in the WHERE clause have to be chosen from the set of possible column headers. Aggregation can have one of the four values I mentioned above.
The WHERE clause consists of one or more conditions. In fact we probably never find more than three conditions in our questions and most of the time it is just one. A condition has three parts: a column name, an operation and a value. The operation between WHERE clause column and value can have these three forms: '=','<' and '>'.

3.3 Methods

In the following sections I will give a detailed description of the computational methods that were used during the practical work. At first, I will explain some of the choices regarding machine translation that I took to come up with the best German training corpus. Then we will have a look at the neural network architecture. I will explain each of the different networks, used to generate the information in the canonical representation. (see listing 3.4)
Finally I introduce the concept of embeddings and present the pre-trained embeddings that I experimented with.

3.3.1 Translation

The quickest way to develop a neural network learning chatbot in a different language, namely German, should be to translate the natural language sentences that are aligned with SQL expressions in the training data. The system is then trained on the translated data again and should ideally already understand the new language. In our case, this means translating all the questions as well as all the headers that belong to them for the WikiSQL data.
More than 80'000 sentences are not realistic to be translated manually by human translators in reasonable time- and cost-efficiency. It is also not necessary since it is possible to use a machine translation system to translate these simple and short sentences. The crucial part with this idea is that these translations are as accurate as possible. This will ensure the quality of the corpus and its actual usefulness as a training resource for a German chatbot.
It is not important that the translations are always perfectly on point, because neural network learning algorithms are generally known to be resistant to some amount of noise in their inputs. So, a good machine translation service will probably serve

a good enough service, while being highly efficient and low cost.
For this translation task we considered two translation services. One was the Google translate API and the other one was DeepL.com. Both of these services use big scale deep learning to train their neural machine translation systems. They also both provide a well-documented and easy to use web API which takes requests and charges a certain fee per translated character. While Google translate is kind of a stable veteran in the machine translation business, DeepL is seen as a runner-up that features some exciting benchmark results.
To find out which of the services works better for our cause, I translated the whole WikiSQL corpus with both of them and evaluated a random sample of both of the outcomes.
First there is to say that, while both of these services provide very easy access for reasonable prices, actually only Google translate's API was ready to use right away. At the time of this writing it was not possible for a private person to access DeepL's web API. The reason was supposedly that they are overwhelmed by requests at the moment.
To have the DeepL translations nonetheless, we decided to use a web scraper that puts the questions as queries into the translation box on the DeepL web page. As soon as the result appears, the bot retrieves it from the result box and puts the next query. This is not a recommended approach, as it can impact the performance of a web page and is not the intended way to use this application. But since our corpus was relatively small and we couldn't get access in a different way, this was the only way to go.
This method takes a lot longer than using an API. To avoid the web server automatically blocking the IP that sends a huge amount of requests, different sorts of timeouts had to be used. To speed up the process I ran the bot from 5 different machines at once, 3 of which were small amazon computing instances. In this manner the 80'000 sentences could be translated in roughly a week.
To evaluate the quality of the translations I randomly picked 200 of them from the corpus. I manually rated them from 1 to 5 based on my subjective judgment of correctness. Results can be found in table 3.1. The quality of the translations is quite good, especially for shorter sentences. It is not often the case that a question is not comprehensible anymore after the translation or completely changes its meaning which would defeat the purpose of the corpus. DeepL sometimes corrupts numerical characters. In the translations we can find examples of numbers where parts were interchanged or cut out. It is very rare though and can be fixed in post-processing. Google translate seems to have problems with detecting named entities which creates a big problem for our corpus, which contains many named entities in column names and values. So if these are not translated correctly, the bot will learn wrong

Table 3.1: Quality assessments of translation services

	DeepL	Google translate
mean	4.70	4.55
median	5.0	5.0
standard deviation	0.48	0.61

associations.
As can be seen in table 3.1. DeepL translations are generally a bit better than the Google translate ones. So it is these translations that we are using for the training. Column names or headers are also part of the input to the system. To make sure that the training process works as intended, all of the header tokens were translated as well. In the corpus we can also find tokens of the WHERE clause values. These are mostly numerical, but sometimes they are words like "color='blue' ". I translated these as well to be sure that we can evaluate the generated German SQL expressions against the ground truth later.

3.3.2 Neural Network Training

In the following I will explain how the different parts of the system are trained to generate an SQL expression from a natural language input. We might have an input like "Who was the winning team of the game that was contested on February 1, 2009?". What we want our system to learn is the variable parts of the SQL expression. As seen in the canonical representation listing 3.4. The resulting SQL is then "SELECT winner WHERE date='February 1, 2009';".
According to this structure, it is three parts that we want to determine. Aggregation, SELECT column and WHERE clause. An aggregation function is not used in this example because we just select one value and aggregations can only be produced from a group of values. The select column is "winner" and the WHERE clause is "date='February 1, 2009' ".

Aggregation To predict an aggregation operation like the count of players from Georgia in the example previously given, first a scalar attention score is computed on all the tokens of the input question, $\alpha_t^{inp} = W^{inp} h_t^{enc}$, where W^{inp} is the weight matrix and h_t^{enc} is the state of the encoder corresponding to the t^{th} word in the

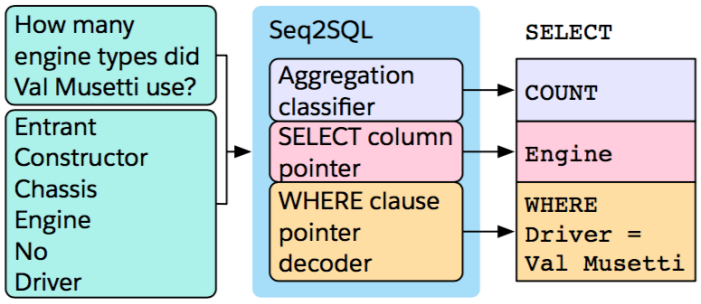

Figure 2: The Seq2SQL system has three components: Aggregation, SELECT column and WHERE clause. On the left there is the input, a question and the headers of the corresponding table. On the right the SQL expression.

input sequence. From these a distribution is produced, $\beta_t = softmax(\alpha^{inp})$. The input representation κ^{agg} is the sum over the input encodings h^{enc} weighted by the normalized scores β^{inp}.

$$\kappa^{agg} = \sum_t \beta_t^{inp} h_t^{enc}$$

Next we compute α^{agg} by applying a multi-layer perceptron to the input representation κ^{agg}. α^{agg} denotes the scores over the subset of the three SQL aggregation functions COUNT, MIN, MAX, and the no-aggregation 'None'.
Then we apply the softmax function to the output of the perceptron to obtain the probability distribution over the set of possible aggregation operations, $\beta^{agg} = softmax(\alpha^{agg})$. This distribution lets us choose the most probable of the four aggregation possibilities.

Select Column The selection column depends on the table columns as well as the question. SELECT column prediction is a matching problem. We have column names like "Player", "Nationality", "Position", etc. encoded with an encoder LSTM network. Given the list of column representations and a question representation, the column is selected that best matches the question.
Similar to the procedure above, column names and input are represented by applying some transformation on them. Namely encoding all the column names with LSTM networks and the same procedure on the input question as for the aggregation.

$$h_{i,j}^c = LSTM(emb(x_{j,t}^c), h_{j,t-1}^c)$$

$h_{j,t}$ denotes the tth encoder state of the jth column. A multilayer perceptron is then applied to the column representations, conditioned on the input sentence representation. It outputs a distribution of scores for all the possible columns, which represents their probability to be the correct column.

Where Clause The WHERE clause is the most complicated structure to predict. It is necessary to predict the subset of columns that is present in the WHERE clause and for each of them we also have to predict an operator and a value.
For example, in the question above, the column is "date", operator is "=" and the value is "February 1, 2009". Attention weights are computed so that the network predicts only the set of columns that are part of the WHERE clause. Taking the weighted sum of the attention weights and the LSTM network hidden layers output of the question encodings, we can compute the probability of the columns to be in the set of WHERE columns.
Once we have these probabilities, we need to decide how many of the columns are in the clause. We use a pointer network again to predict the number of WHERE clause constraints. Then we choose the number of columns with the highest probability scores.
Predicting the operation between column and value is a three-way classification problem. The model needs to choose between '<','>' and '='. The pointer network uses column attention again to capture the dependency between column and operator slot.
The value slot is predicted with a sequence-to-sequence model. The value is generally a substring of the natural language question. The sentence token embeddings are encoded with a bi-directional LSTM network as before. The decoder network is another pointer network. Generally we compute the probability that the next token of the value string is the i-th token in the natural language question.

$$P_{val}(i|Q, col, h) = softmax(a(h))$$

where h is the hidden state of the previously generated sequence, Q is the input question, *col* is the column that is associated with the value and $a(h)$ is the decoder output of each of the tokens in the input question. Note that an <END> token is added to the input tokens. The model takes all input tokens with probabilities higher than the <END> token as part of the value string. Input tokens with probabilities lower than the <END> token do not belong to the value string.

3.3.3 Embeddings

Embeddings are n-dimensional vectors, where the numbers in the vector are probabilities of the word to appear with a certain context word. These vectors represent a word based on its surroundings in natural language. Embeddings are learned and updated from examples of words and context in natural language text sources. A neural network is used to predict probability of context based on words if the Skip-gram algorithm is the architectural choice. If we calculate the probability of words given a context, they are produced with Continuous Bag of Words (CBOW) algorithm. The choice of algorithm brings differences in the properties of the output embedding table. The Skip-gram approach tends to produce embeddings that capture more semantic details and can also represent different concepts of the same word.[2]

Embeddings are ideally extracted from very big text corpora. Words become points in space and much more meaningful input to neural networks. If the vocabulary in the data is rather domain specific, it makes sense to train embeddings on domain specific text. If not, there are already pre-trained embeddings available. They are extracted from corpora like Wikipedia or Europarl. [Koehn, 2005]

To train the networks in my experiments, I used different types and sizes of pre-trained word embeddings. They were trained using a Word2Vec [Mikolov et al., 2013] approach which uses a combination of Skip-gram and CBOW networks. First I used polyglot embeddings. [Al-Rfou et al., 2013] These were some of the first pre-trained embeddings available. They are trained for 110 languages on their respective Wikipedia corpus. The embedding files are very small for the polyglot embeddings, since they only consist of 64 dimensional vectors for the 100'000 most frequent words of each language. That means we have 100'000 words to look up and 64 context probability values associated with each of them.

These embeddings were useful to get a first baseline and run a quick training. After that I tried bigger embedding files. I used pre-trained vectors from fastText. [Bojanowski et al., 2016] They are also trained on German Wikipedia but featuring a vocabulary of more than 2 million words. These vectors have 300 dimensions.

It is possible to simultaneously continue updating the probabilities of the pre-trained word embeddings while running the Seq2SQL training algorithm. The probabilities are adjusted based on the context in which the word is found in the training data. In addition to that it is possible to initialize words that are not found in the pre-trained embedding file with a zero embedding. This embedding can then be updated between the training iterations as well, so that we produce some of our own embed-

[2] https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/ - 25.06.2018

dings on our data.
One way to dramatically increase the training data size is to use multilingual embeddings.[Conneau et al., 2017] These embeddings are taken from word-aligned parallel corpora of different languages which allows us to compute probabilities of words to be in context with words of different languages. We take the words that are aligned to the context words of the word that we are looking at. An English word like "house" might be in context with different English words. We can look at the alignments of the context words of the German "Haus" to English to compute the probabilities for the same context words in another language.
In the end we achieve vectors for different languages which all use the same vector space. They feature probabilities of the same words being in context with them. With these vectors it is possible to use training material of different languages, because the multilingual words that are semantically quite equivalent will be very close to each other in the vector space. The language of the words does not matter anymore because they are represented by their surroundings which are aligned across languages.

3.4 Evaluation

To evaluate the quality of the trained system we took the splits of the WikiSQL dataset of the Seq2SQL project. The whole dataset is partitioned into 56'355 sentences for training, 8'421 for development and 15'878 for testing. With the training set I trained models for aggregation, SELECT column and WHERE clauses. Due to the different architectures, Aggregation and SELECT column were trained separately from the WHERE clauses. Aggregation and SELECT column prediction use the same type of network which just outputs one column name and an aggregation to it. For the WHERE clause a pointer network is used and the output consists of sequences of tokens of variable length.
First of all, I wanted to find out what the impact of translating all the table headers to German was on the quality of the model. My hypothesis is that, since we are using embeddings to encode the input tokens, it should not depend on the language too much. All the inputs to the system are encoded to a network representation which is then decoded to attain a probability distribution over the headers. The connection between German words in the question and English headers can be learned by a neural network in the same way it can with German headers.
To compare and contrast different starting data and model parameters, I ran the training algorithm several times. To see the details of the different settings, please

refer to table 3.3.

	Aggregation	SELECT column	WHERE column
setting 1	**0.89**	0.79	0.33
setting 2	0.87	0.82	0.09
setting 3	0.87	0.80	0.31
setting 4	0.87	**0.83**	0.31
setting 5	-	-	0.32
setting 6	-	-	0.29
setting 7	-	-	0.14
setting 8	-	-	**0.33**
Seq2SQL reference	0.901	0.889	0.601
English baseline	0.872	0.865	0.005

Table 3.2: Accuracy results of different training configurations on the test set

For setting 1 I used the small polyglot embeddings pre-trained on Wikipedia (see section 3.3.3), the input questions translated by DeepL and the original English headers. The flag for retraining embeddings during training was set to false. I trained the aggregation and SELECT clause model for 5 epochs and the pointer network for the WHERE clause for 50 epochs. These different numbers are results of the WHERE clause training algorithm taking less time to finish an epoch.
With this configuration, we get the training results that can be seen in table 3.2 in the first row. Interestingly the aggregation was predicted the most accurate in this first trial out of all the trained models. The SELECT column is not as accurately chosen from the possible column headers. The WHERE clause prediction does not work very well on German training data. We are speaking of 50% decrease in accuracy compared to the English version of the Seq2SQL experiments.[Xu et al., 2017]
Aggregation is nearly as accurate as the Seq2SQL results at least in the first model. The SELECT column sees about a 10% drop in accuracy which is still acceptable, considering that in the original experiment the questions were tailored to the header columns. In the creation of the corpus the sentences were built looking at the correct SQL expression already. (See section 3.2.1) It was to be expected that this prediction is a bit less accurate since after translation we often do not find the header tokens in the question anymore. It also happens when we translate the headers, because often a word is translated differently in context of the question than separately.

	Translated Header	Translated Values	Embedding	Continue training embeddings	Epochs
setting 1			polyglot		agg: 5, ptr: 50
setting 2	X		polyglot		agg: 5, ptr: 50
setting 3			fastText		agg: ∼10, ptr: ∼100
setting 4			fastText	X	agg: ∼10, ptr: ∼100
setting 5	X	X	fastText	X	∼100
setting 6	X	X	fastText+GloVe	X	∼100
setting 7		X	fastText	X	∼100
setting 8	X	X	MUSE multilingual embeddings	X	∼100

Table 3.3: Differences in training parameters during model training

What happens when we take the translated headers instead of the original ones during the training, can be seen with setting 2. We can easily compare setting 2 and setting 1. They both use the same settings the only difference being the column headers translated or left original. While the SELECT column prediction is a lot better than in setting 1, aggregation is worse and the pointer network does not produce much usable output anymore.

One obvious way to boost the performance of the system is to use bigger pre-trained embeddings to lookup the tokens. In setting 4 I used large embedding files with 2'000'000+ words. I used the training file with English headers and German translated questions again. This time the 300 dimensional pre-trained embeddings from fastText were used to encode the input. The fact that these embeddings were trained on Wikipedia makes them very suitable for the task, since the questions are also constructed on Wikipedia data.

Setting 4 got 83% for the SELECT column prediction which is the best result I could achieve. It is a high number considering that there were only English headers to choose from, based on German questions. In fact it is even a bit better than when we use the translated headers as can be seen in table 3.2.

After some discussions, we had identified one possible source for the practically zero results of the pointer network in WHERE clause prediction with translated German

headers (setting 2 table 3.2). The problem was in evaluation where we are comparing the output of the pointer network to the ground truth.
The WHERE clause values in the ground truth are normally a subset of tokens of the input question and header tokens. The network uses a sequence to sequence transformation approach to find the value based on the input question and the headers. So on German input questions and German headers the value will often very much resemble German language and not compare well to the English values in the ground truth.
To overcome this problem I translated all the values in our ground truth. I only substituted them if, after translation, the German value could be found in the input question as well. Otherwise this would probably have been rather detrimental to the task.
I trained just the WHERE clause prediction again on the data with translated values. It helped and increased the accuracy from 0.09 in setting 2 to 0.32 in setting 5. In setting 7 I tried using English headers with translated WHERE clause values. The result confirms that the values are also generated from headers as well as the questions and that the evaluation problem has indeed been caused by the difference in language of the WHERE clause values.
In settings 6 and 8 I tried different embeddings again.
In setting 8 I used a combined corpus of all German sentences and all original English sentences to train the system. The tokens were represented by German embeddings and English embeddings which were produced in a multilingual manner, as explained in the embeddings section. [Conneau et al., 2017]

3.5 Limitations

As I have already mentioned at several points in this thesis, the approach discussed here to build a German question-answering chatbot, has some limitations.
We have seen that the WHERE clause prediction accuracy couldn't be trained to a quality that can be compared to the one in the original Seq2SQL experiments. The biggest reason for this is that the corpus has been built very artificially and the question sentences were often quite similar to the SQL that they were derived from. With the translation to German this similarity has vanished in many cases. Especially since the English sentences were often ungrammatical and not making sense anyways. This often led to nonsensical translations.
When column headers and WHERE clause values were translated, they were often translated by a different word than what could be found in the sentence, which

makes it harder for the algorithm to associate the correct columns and values with an input question. If we consider an example like “Who won on September, 1?” with the automatic translation “Wer gewann am ersten September?”. Our model might even predict the WHERE clause value to be “ersten September”, but then in evaluation we compare to the translated groundtruth value and it could be “1. September”. We will not have a match in this case.
These were the limitations that keep us from achieving high accuracy values in test set evaluation. In general the task can also be said to be too simple for a real world scenario. Even if the model would have predicted very well on the test set, it is still not able to form more complex SQL queries and answer questions that are out of the training domain reliably. The concept is no sufficient proof that a customer care chatbot could be created in a similar way. For this, domain specific training corpora and real SQL queries are necessary.

4 Conclusion

In the beginning of this thesis I gave an overview of rule-based chatbot systems and machine learning approaches. For the task of building an information retrieval chatbot, I tested a sequence to sequence neural network algorithm. The task was that we want to predict a complex structure, namely an SQL expression, from an input question and the headers of the table which contains the information. The resulting SQL query is heavily dependent on the inputs. On the other hand it is a far abstraction from the original sentence and it does not always use the elements that can exactly be found in the words of the natural language sentence.
We have to draw a connection between the natural language words and the known column headers. Generating the correct SQL expression is mainly a question of whether we can find a function that is able to map a subset of the predefined table headers to an input sentence. Neural networks are a great way to find a function like this.
Another big advantage compared to a rule based system is that if we train the system on enough data, it will be able to generalize to big amounts of unseen data without having to readjust constantly to new input like in a rule based system. Out of these reasons this approach seemed the most reasonable.
I could show with my project that adapting such a chatbot to another language can be quite challenging even if the approach seems easy and straight-forward. Simply translating the data with a machine translation service and then retraining it, resulted in a system that can predict 33% of the SQL queries correctly. That is significantly lower than the original English system at 60%.
I reached accuracy values of 5-10% lower than the English system in the prediction of aggregation and SELECT column, which are the simpler parts of the SQL expression. On the more complex part of the WHERE clause generation, the system still produces a sensible output, but compared to the English system it is loosing accuracy. I tried different possible solutions for the WHERE clause problem. In the end none of them turned out to improve the result significantly.
To increase performance of the German chatbot to a comparable level with the English one, different training data would be needed. The WikiSQL data set is too closely derived from the SQL expression that are used to evaluate the performance of

the model, so in the original language the higher accuracy is much easier to achieve. To give an ultimate answer whether a rule-based or a machine learning system is a better fit for this task, it would be very interesting to adapt a rule-based chatbot to German language.

References

S. A. Abdul-Kader and D. J. Woods. Survey on chatbot design techniques in speech conversation systems. *International Journal of Advanced Computer Science and Applications*, 6(7), 2015. doi: 10.14569/IJACSA.2015.060712. URL http://dx.doi.org/10.14569/IJACSA.2015.060712.

R. Al-Rfou, B. Perozzi, and S. Skiena. Polyglot: Distributed word representations for multilingual nlp. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 183–192, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W13-3520.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

J. Bornholt, E. Torlak, D. Grossman, and L. Ceze. Optimizing synthesis with metasketches. *SIGPLAN Not.*, 51(1):775–788, Jan. 2016. ISSN 0362-1340. doi: 10.1145/2914770.2837666. URL http://doi.acm.org/10.1145/2914770.2837666.

D. Britz. Deep learning for chatbots. Website, 2016. URL http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/. Last time retrieved 30.06.2018.

A. Conneau, G. Lample, M. Ranzato, L. Denoyer, and H. Jégou. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017.

J. Inscho. Literature review: Chatbots & conversational experiences. Website, 2017. URL https://studio.carnegiemuseums.org/literature-review-chatbots-conversational-experiences-566de218f92a. Last time retrieved 30.06.2018.

D. Jurafsky and J. H. Martin. *Speech and Language Processing*. Prentice Hall, 2nd edition, 2009.

P. Koehn. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand, 2005. AAMT, AAMT. URL http://mt-archive.info/MTS-2005-Koehn.pdf.

M. L. Mauldin. Chatterbots, tinymuds, and the Turing test: Entering the Loebner prize competition. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, pages 16–21, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence. ISBN 0-262-61102-3. URL http://dl.acm.org/citation.cfm?id=199288.199285.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL http://dblp.uni-trier.de/db/journals/corr/corr1301.html#abs-1301-3781.

D. Mladenić and L. Bradeško. A survey of chabot system through a Loebner prize competition. 2012.

J. v. Neumann. *The Computer and the Brain*. Yale University Press, New Haven, CT, USA, 1958. ISBN 0300007930.

M. J. Pereira and C. Luisa. Just.chat - a platform for processing information to be used in chatbots.

K. Scott. Popular use cases for chatbots. Website, 2016. URL https://chatbotsmagazine.com/popular-use-cases-for-chatbots-925ef8f2b48b. Last time retrieved 30.06.2018.

B. A. Shawar and E. Atwell. Chatbots: Are they really useful? *LDV Forum*, 22: 29–49, 2007.

P. Surmenok. Chatbot architecture. Website, 2016. URL https://medium.com/@surmenok/chatbot-architecture-496f5bf820ed. Last time retrieved 30.06.2018.

A. M. Turing. Computing machinery and intelligence. 59:433–460, 1950.

O. Vinyals and Q. V. Le. A neural conversational model. *CoRR*, abs/1506.05869, 2015. URL http://arxiv.org/abs/1506.05869.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5866-pointer-networks.pdf.

J. Weizenbaum. ELIZA - a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, Jan. 1966. ISSN 0001-0782. doi: 10.1145/365153.365168. URL http://doi.acm.org/10.1145/365153.365168.

X. Xu, C. Liu, and D. Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR*, abs/1711.04436, 2017. URL http://arxiv.org/abs/1711.04436.

V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017. URL http://arxiv.org/abs/1709.00103.