

CZECH TECHNICAL UNIVERSITY IN PRAGUE  
FACULTY OF ELECTRICAL ENGINEERING  
DEPARTMENT OF COMPUTER SCIENCE



## MASTER THESIS

**Creation of Dialog Applications Based on  
Schema.org Ontology**

Bc. Tatiana Okonechnikova

Supervisor: Ing. Jan Šedivý, CSc.

January 2019



## **Author Statement**

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 1<sup>st</sup> January 2019

.....  
signature



## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Okonechnikova** Jméno: **Tatiana** Osobní číslo: **393194**  
Fakulta/ústav: **Fakulta elektrotechnická**  
Zadávající katedra/ústav: **Katedra počítačů**  
Studijní program: **Otevřená informatika**  
Studijní obor: **Datové vědy**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Tvorba dialogových aplikací na základě ontologie schema.org**

Název diplomové práce anglicky:

**Creation of dialog applications based on schema.org ontology**

Pokyny pro vypracování:

Review the conversational AI technologies technologies. Focus on a semi-automatic creation of a dialog from structured and semi-structured data. As a source of structured data use the RDF knowledge database (DBpedia, Wikidata). Use a straightforward static web page as a representant of the semi-structured data. Design and implement semi-automatic application creating a simple dialog for intelligent speakers, such as Amazon Alexa or Google Home. Select a topic, (for example movies) find the RDF database or a web portal and build and test simple dialogs conveying user selected information. Test the application with at least three users and report the results.

Seznam doporučené literatury:

Speech and Language Processing <https://web.stanford.edu/~jurafsky/slp3/> (vybrané kapitoly)

Jméno a pracoviště vedoucí(ho) diplomové práce:

**Ing. Jan Šedivý, CSc., velká data a cloud computing CIIRC**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **02.02.2018**

Termín odevzdání diplomové práce: **08.01.2019**

Platnost zadání diplomové práce: **30.09.2019**

\_\_\_\_\_  
Ing. Jan Šedivý, CSc.  
podpis vedoucí(ho) práce

\_\_\_\_\_  
podpis vedoucí(ho) ústavu/katedry

\_\_\_\_\_  
prof. Ing. Pavel Ripka, CSc.  
podpis děkana(ky)

## III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

\_\_\_\_\_  
Datum převzetí zadání

\_\_\_\_\_  
Podpis studentky

## *Abstrakt*

Cílem této práce je navrhnout a naimplementovat dialogovou aplikaci na základě ontologie Schema.org. Aplikace je schopna vést jednoduchý dialog s uživatelem, který je založen na poloautomatickém přístupu. Chatbot se zakládá na ontologii Schema.org, pomocí které je schopen vyčítat strukturovaná i polostrukturovaná data z různých datových zdrojů, kterými jsou RDF databáze, respektive webové stránky. Ontologie je také využita chatbotem pro aktivní orientaci ve zvolené doméně, která je zaměřena na filmy. Pro implementaci chatbota v programovacím jazyce Python využíváme známé NLP algoritmy pro extrakci dat z uživatelského vstupu. Provedli jsme také integraci námi vytvořeného chatbota do virtuálního agenta Amazon Alexa a do IM platformy Slack. Chatbota vyhodnocujeme na základě reálných konverzací s uživateli.

Klíčová slova: dialog, chatbot, ontologie, NLP, poloautomatické vytvoření, strukturovaná data, polostrukturovaná data, extrakce dat

## *Abstract*

This thesis aims to design and implement dialog application based on Schema.org ontology. The application uses a semi-automatic creation approach to be able to conduct a simple dialog with a user. Schema.org ontology helps the chatbot to read structured and semi-structured data from different data sources, namely RDF databases and web pages. Chatbot also uses the ontology for active orientation in the given domain which is in our case focused on movies. For the implementation of the chatbot in Python we utilized widespread NLP algorithms for data extraction from a user utterance. Moreover, we have integrated the chatbot into the smart speaker Amazon Alexa and the IM platform Slack. The chatbot is evaluated by real dialogs with users.

Keywords: dialog, chatbot, ontology, NLP, semi-automatic creation, structured data, semi-structured data, data extraction

### *Acknowledgements*

I would like to thank my supervisor Ing. Jan Šedivý, CSc. for our consultations and his advice. I also highly appreciate my family's support.





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Thesis Structure . . . . .	2
<b>2</b>	<b>Chatbots and Virtual Assistants</b>	<b>3</b>
2.1	Chatbots Classification . . . . .	4
2.1.1	Purpose of Maintaining a Conversation . . . . .	4
2.1.2	Response Generation . . . . .	5
2.1.3	Capability of Information Providing . . . . .	6
2.2	Ontology-based Chatbots . . . . .	6
<b>3</b>	<b>Knowledge Organization</b>	<b>8</b>
3.1	Ontologies . . . . .	8
3.2	Schema.org . . . . .	11
3.3	Knowledge Graph . . . . .	11
3.4	RDF Triple . . . . .	11
3.4.1	RDF Store . . . . .	13
3.5	SPARQL . . . . .	14
<b>4</b>	<b>Natural Language Processing</b>	<b>18</b>
4.1	Natural Language Understanding . . . . .	18
4.1.1	Intent Classification . . . . .	18
4.1.2	Named-entity Recognition . . . . .	19
4.1.3	Anaphora Resolution . . . . .	21
4.2	Natural Language Generation . . . . .	21

<b>5</b>	<b>Machine Learning Algorithms for NLP</b>	<b>24</b>
5.1	Training of Models and Evaluation . . . . .	24
5.1.1	Training, Validation, and Testing Datasets . . . . .	26
5.1.2	Cross-validation . . . . .	26
5.2	Artificial Neural Networks . . . . .	27
5.2.1	Feedforward Neural Network . . . . .	27
5.2.2	Recurrent Neural Networks . . . . .	32
5.3	Word Embedding . . . . .	34
5.3.1	Word2vec . . . . .	35
5.3.2	GloVe . . . . .	36
5.3.3	FastText . . . . .	37
5.4	Multi-armed Bandit problem . . . . .	37
5.4.1	Strategies . . . . .	38
<b>6</b>	<b>Implementation</b>	<b>40</b>
6.1	Architecture Overview . . . . .	40
6.2	Single Building Blocks Description . . . . .	42
6.2.1	Intent Classification Module . . . . .	42
6.2.2	Information Extraction Module . . . . .	43
6.2.3	Context Module . . . . .	51
6.2.4	Knowledge Base Module . . . . .	52
6.2.5	Attribute Choice Module . . . . .	56
6.2.6	Answer Generating Module . . . . .	58
6.3	Implementation Details . . . . .	61
6.3.1	Integration to Other Systems . . . . .	62
6.3.2	Keras . . . . .	64
6.4	Datasets Description . . . . .	64
<b>7</b>	<b>Experiments</b>	<b>66</b>
7.1	NER comparison . . . . .	66
7.1.1	Train and Test Setting . . . . .	66
7.1.2	Results . . . . .	68
7.1.3	Runtime . . . . .	70
7.2	Intent Classification . . . . .	70
7.3	Simple Dialog Examples . . . . .	79

7.3.1	Dialog 1 . . . . .	80
7.3.2	Dialog 2 . . . . .	82
7.3.3	Dialog 3 . . . . .	83
7.3.4	Dialog 4 . . . . .	84
7.3.5	Dialog 5 . . . . .	85
7.3.6	Summary . . . . .	87
<b>8</b>	<b>Conclusion</b>	<b>88</b>
<b>A</b>	<b>Attached Files</b>	<b>94</b>

# List of Tables

3.1	SPARQL query results returned for (3.2).	16
3.2	SPARQL query results returned for (3.3).	17
5.1	Notation of the used symbols in equations (5.25)-(5.30)	33
5.2	Examples of preserved relationships.	35
6.1	Single parts of Amazon URL	54
6.2	Table with examples of production rules $\alpha \rightarrow \beta_1 \dots \beta_n$ .	60
7.1	Overview of characteristics of the training data for Stanford NER (for each fold).	67
7.2	Overview of characteristics of the given lists of data to the trie.	67
7.3	Overview of characteristics of the testing data (for each fold).	68
7.4	Comparison of two methods for NER by their performance.	68
7.5	Comparison of two methods for NER by their average runtime per tagged sentence.	70
7.6	Sizes of datasets for NN training and evaluation.	71
7.7	Results on simple LSTM cells depicted in Figure 7.1.	72
7.8	Results on LSTM cells followed by dense layer depicted in Figure 7.2.	72
7.9	Results on architecture depicted in Figure 7.3.	72
7.10	Results on architecture depicted in Figure 7.4.	72
7.11	Results on architecture depicted in Figure 7.5.	73
A.1	Description of attached files	94

# List of Figures

2.1	Google Home and Amazon Echo smart speakers. . . . .	4
3.1	Movie class hierarchy. . . . .	9
3.2	Movie ontology graph. . . . .	10
3.3	Amazon Neptune Knowledge Graph based on different widespread ontologies. . . . .	12
3.4	RDF graph. . . . .	13
3.5	Movie RDF graph based on knowledge taken from DBpedia. . . . .	14
5.1	Block schema of LSTM cell. . . . .	34
6.1	Architecture Overview. . . . .	41
6.2	Movie titles stored in trie structure. . . . .	48
6.3	Amazon Alexa Integration Schema. . . . .	63
7.1	Simple LSTM cell variant. . . . .	73
7.2	Shallow RNN with dense layer. . . . .	74
7.3	RNN with two dense layers. . . . .	75
7.4	RNN with three dense layers. . . . .	76
7.5	RNN with four dense layers. . . . .	77
7.6	Accuracy of chosen models on a validation set during training. . . . .	78
7.7	Accuracy of chosen models on a training set during training. . . . .	79

# Listings

6.1	HTML snippet for the movie title . . . . .	55
6.2	HTML snippet for the movie price on DVD . . . . .	55

## Used notation

Symbol	Meaning
$\Sigma$	set of characters in a language
$\Sigma^+$	set of all sequences of letters from $\Sigma$ (corresponds to Kleene plus)
$x^T y$	dot product of vectors $x, y \in \mathbb{R}^n$
$D_C(x, y)$	cosine distance of vectors $x, y \in \mathbb{R}^n$ , $D_C(x, y) = 1 - \frac{x^T y}{\ x\ _2 \ y\ _2}$
$\{a_i\}_{i=1}^n$	finite sequence with elements $a_1, \dots, a_n$
$\{a_i\}_{i=1}^\infty$	infinite sequence with elements $a_i$
$s[i : j]$	substring of string $s$ from $i$ -th letter to $j$ -th letter
$s[i : ]$	substring of string $s$ from $i$ -th letter to the end
$ s $	length of string $s$
$L(s, s')$	Levenshtein distance of two strings $s, s'$
$\mathbb{N}_0$	set of non-negative integers
$x \odot y$	Hadamard product of vectors $x, y$

## Used abbreviations

<b>Abbreviation</b>	<b>Full name</b>
AGI	Artificial General Intelligence
AI	Artificial Intelligence
API	Application Programming Interface
ASIN	Amazon Standard Identification Number
AWS	Amazon Web Services
CBOW	Continuous Bag-of-words
CRF	Conditional Random Field
DFA	Deterministic Finite Automaton
DVD	Digital Video Disc
ELU	Exponential Linear Unit
GloVe	Global Vectors
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IM	Instant Messaging
IMDb	Internet Movie Database
IRI	Internationalized Resource Identifier
IT	Information Technology
JSON	JavaScript Object Notation
KOS	Knowledge Organization Systems
LSTM	Long Short-term Memory
ML	Machine Learning
NER	Named-entity Recognition
NFA	Non-deterministic Finite Automaton
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
NN	Neural Network



NoSQL	Non SQL
OWL	Web Ontology Language
PCFG	Probabilistic Context-Free Grammar
POMDP	Partially Observable Markov Decision Process
RDF	Resource Description Framework
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SELU	Scaled ELU
SMBO	Sequential Model-Based Global Optimization
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SVM	Support Vector Machine
Turtle	Terse RDF Triple Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XPath	XML Path Language



# Chapter 1

## Introduction

### 1.1 Motivation

Creating dialog applications is starting to be a more popular problem in the AI world. A dialog stands for interactive conversational communication between a human on the one side and an artificial system on the other side. A dialog application means a computer program which is able to imitate human behavior while a conversation is being conducted.

Already in the second half of the twentieth-century intelligent computer programs were beginning to be invented. One of the first conversational computer assistants based on pattern matching rules to statements, well-known as ELIZA, was invented by J. Weizenbaum in the latest 1960s. The program was successful in the community of people with mental illness because it simulated a Rogerian psychotherapist<sup>1</sup> and the users had a recovery effect.

Then, other similar conversational agents started to appear. Another name for such computer program assistant is *a chatterbot* or just *a chatbot*. The significant idea of these conversational systems is that a computer is trying to be as intelligent as a human and to understand the human language. And the crucial long-term aim of such thinking machines is to help humans in any field of life or just to bring entertainment into their everyday moments.

The goal of the thesis is to create an entertainment chatbot which would be able to lead a conversation with a user in a specific domain. To build such chatbot, we utilize the concept of ontology, which defines entities with their properties and different relations between them, to provide the semantics of domain information. We consider domain-specific knowledge graph based on Schema.org ontology. The knowledge graph could help the chatbot to grasp the conversation topic and to be capable of driving a dialog with a user.

---

<sup>1</sup>Person-centered therapy

The chatbot also has to deal with multiple NLP tasks. Tasks such as intent classification and named entity recognition fall into NLU, whereas the task of answer generation comes under NLG, where both NLU and NLG are subtopics of NLP. Moreover, Yampolskiy (2013) states that NLU is considered to be AI-hard, because to solve it means to create AGI, respectively, to make computers think like humans.

To solve, e.g., the intent classification task it is needed to apply algorithms from ML, specifically, classification algorithms. Such ML algorithms are a type of supervised learning which means it requires a set of observations along with their ground-truth annotations. Obtaining appropriate domain-specific training datasets, which are large enough and capture rich dispersion of a natural language, is a frequently faced issue in ML.

## 1.2 Thesis Structure

This thesis is organized into three main parts. The first part includes Chapter 2, Chapter 3, Chapter 4, Chapter 5 and focuses on conversational AI technologies. The second part includes Chapter 6 and describes chosen approach and its implementation. And the last part includes Chapter 7 in which experiments are performed and evaluates the overall results.

## Chapter 2

# Chatbots and Virtual Assistants

We are living in an era of information technology evolution. The development of robots or personal assistants is gaining momentum. The purpose of it is an effort to automatize more things in our daily life.

Some years ago, we started quite frequently to hear the term "smart home." The word means intelligent systems which were invented to secure home comfort and to make daily human life easier. As Alam et al. (2012) mentioned, these systems are divided into three main categories: comfort services (e.g., controlling home appliances and devices), healthcare services (e.g., identifying user health conditions, ensuring assistive services, or generating local warnings if required) and security services (e.g., providing authorized access control to avoid security threats). These smart systems customarily can respond to a user's voice or just to a button click. But the functionality is restricted, so voice commands can only correspond to either activating or deactivating of home appliances or devices. Also, a button click is similarly limited to turn-on and turn-off toggling. As a consequence, we can conclude that such smart home assistants help their users by taking over control of some domestic things.

After a few years, virtual assistants have appeared on the IT market. It is a software agent which is inbuilt into smart devices. The purpose of these assistants is to perform simple user tasks such as daily individual activity scheduling. These intelligent personal helpers can receive user's commands given by voice user interface or text user interface. Devices with voice user interface are commonly named smart speakers. As an instance of voice-controlled devices, shown in Figure 2.1, can be mentioned Amazon Echo which is driven by Amazon Alexa virtual assistant and Google Home which is managed by an intelligent personal agent called Google Assistant. However, the majority of other existing virtual assistants uses rather text user interface to have an online chat based on IM technology.

Virtual assistants that use textual user interface are frequently called *chatbots*. The bots are represented by a computer program which has strong NLU capabilities. As far as their aims are concerned, different chatbots have miscellaneous purposes.



Figure 2.1: Google Home and Amazon Echo smart speakers.<sup>1</sup>

However, the most relevant goal of a chatbot is to conduct a conversation with a user and try to understand user's requests.

Now let's take a look at the specific classification of such chatbots.

## 2.1 Chatbots Classification

We would like to start the section with a citation which describes the actual situation in developing chatbots. Levesque (2017) states that current state-of-the-art systems are still a long way from being able to have coherent, natural conversations with humans.

Britz (2016) points out that chatbots can be divided, firstly, according to the purpose of maintaining a conversation with a user, secondly, according to the response generation and, finally, according to their capability of information orientation.

### 2.1.1 Purpose of Maintaining a Conversation

Such conversational agents are divided into two subsequent classes a task-oriented bot and an entertainment-oriented bot, so-called chatbots.

Task-oriented bots are designed for performing particular tasks from users or helping users to achieve a certain goal. How Jurafsky and Martin (2014) mention,

---

<sup>1</sup>Taken from <https://www.xda-developers.com/google-home-amazon-echo-smart-speaker-market-share/>

such agents have short conversations to get information from the user to help complete the task, such as to book a flight, to buy a train ticket, to find a restaurant, etc.

Ilievski (2018) mentions that the current options of implementing a goal-oriented chatbot can be divided into two approaches. The first one is called a fully-supervised approach, in which recurrent neural networks are utilized. Such models require a lot of annotated dialogs. For example, Bordes and Weston (2016) use a Memory Network to solve goal-oriented dialog tasks, where such a network is utilized for multiple tasks. Another example is given in Wen et al. (2016), where an end-to-end dialog system is also implemented by a combination of multiple neural networks.

The second approach for goal-oriented dialog systems is reinforcement learning. Reinforcement learning methods are overviewed in Young et al. (2013). They use POMDPs, in which the hidden state is the actual state of the dialog. It is not directly observable due to the ambiguity of natural language. Also, a policy is learned that decides which action should be taken. Like the fully-supervised approach, there is again a significant obstacle in obtaining the dialog data.

The disadvantage of these approaches is described by Serban et al. (2015), who states that there is a lack of high quality, open-ended, freely available conversational datasets.

Entertainment-oriented bots are designed for providing people enjoyment in the way to have an appropriate conversation. In Jurafsky and Martin (2014)'s book we can find the additional information for such chatbots that they support extended conversations, often have entertainment value and mimic the unstructured conversational nature of human-human interaction. Furthermore, such bots are able to keep the conversation interesting for the person they are chatting with. In addition, (Venkatesh et al., 2018) defines chatbots as one specific type of conversational interface with no explicit goal other than engaging the other party in an interesting or enjoyable conversation.

## 2.1.2 Response Generation

For creating generative chatbots, sequence-to-sequence networks are usually used, which requires a lot of training data. As stated in Kim et al. (2018), such large data sources are today available only for English and Chinese. Even though, the issue of the generative methods is that there is less control over the outputs, so they have more issues with following the rules of grammar when compared with rule-based methods. Tammewar et al. (2017) tried to combine these approaches in a hybrid model to make the final model more robust.

Retrieval-based conversational agents use a repository of predefined responses and some kind of heuristic to pick an appropriate response based on the input and

context. The heuristic could be as simple as a rule-based expression match, or as complex as an ensemble of Machine Learning classifiers. These systems pick a response from a fixed set. Pattern matching involves using regex to find patterns in the incoming text, and classify it into different intents.

### 2.1.3 Capability of Information Providing

Open domain chatbots, sometimes called horizontal, are trying to maintain the conversation that has no thematic limits and can go in all possible directions. The chatbot is destined to retrieve all sorts of information. Such systems don't try to achieve any goal. But a large amount of world knowledge is required that the chatbot returns reasonable responses.

Closed domain chatbots, sometimes called vertical, have restricted conversation to a specific domain, such systems are customarily trying to achieve a specific goal. The chatbot is able to produce responses due to a specific domain but is not required to handle all possible cases, so the user does not expect it to.

## 2.2 Ontology-based Chatbots

The majority of ontology<sup>2</sup>-based chatbots are domain-specific. Altinok (2018) confirms it, an ontology-based conversation is indeed a way of domain-driven conversation. Venkatesh et al. (2018) supposes that a domain-driven conversation agent may be more akin to goal-directed conversations, where the output response space is bounded.

The relevant idea of using an ontology is that using knowledge helps a chatbot drive a conversation in the sense of an ontology helps a chatbot to understand in which direction to continue a conversation with a user and create the appropriate response.

The widespread example of ontology-based chatbots appears on the web pages of different business companies to help the customers obtain desired information which is restricted to the specific company domain.

Altinok (2018) describes an ontology-based chatbot for the banking and finance domain. Its novel approach uses the ontology to keep the context of the dialog. The used ontology represents banking products, their attributes, and relationships that the customer can query. The approach to keeping the current context is based on the fact that the ontology graph can be viewed as a rooted tree in this case – the products are organized hierarchically so that each one points to its subproducts and vice versa. Each element can also have its properties that the user can query too. Now, if the

---

<sup>2</sup>The term "ontology" is described in detail in Section 3.1.



user wants to know something about a product, its name is extracted from the query, and a pointer is set to the corresponding node in the ontology. If the user wants to ask additional information about the previously mentioned product, the pointer keeps the context, and the chatbot, therefore, knows where to search for such information. If the product has multiple sub-products (such as types of accounts or mortgages), the chatbot simply queries the ontology to list the corresponding children nodes so that the user can choose a sub-product or ask for its attributes (such as interest rate). If the user does not like a particular sub-product, the bot traverses the ontology to find a sibling node that was not visited yet – already visited nodes are marked so. This allows the bot to systematically offer the customer alternatives that were not mentioned before. This approach seems quite straightforward and promising for domains whose data structures can be viewed hierarchically (i.e., as a rooted tree) and does not have too much complexity. We will later in this thesis try to generalize this procedure for domains with arbitrarily dense graphs with cycles.

# Chapter 3

## Knowledge Organization

The term means to organize information into schemes to provide consequential information management. It also ensures to gain structured data from unstructured data. This thought is important to a concept of machine-readable data. The purpose of organizing knowledge is to make possible a computer to process such data.

There is an approach to organize the knowledge with classification schemes. A classification scheme enables to arrange objects into the organization, frequently having hierarchical form, structure. Such a structure can be represented by classes and groups of classes.

The widespread example of a classification scheme which organizes knowledge information is an ontology.

### 3.1 Ontologies

One of the first definitions of an ontology is given by Gruber (1993), who defines it as "an explicit formal specification of the terms in the domain and relations among them." Less formally we can describe an ontology as "each element from some existing domain can be described by super-classes/classes/sub-classes/instances. These classes have its properties/attributes. And finally, all classes are interconnected with each other, which creates classes relationships."

The significant purpose of an ontology is to help to convey the meaning of data among computers and also among people.

An ontology is applied to the specific domain and tries to describe this domain using classes, groups of classes and relations between them. A class<sup>1</sup> represents a set of entities or 'things' within a specific domain and a relation<sup>2</sup> describes the interactions between classes or a class's properties.

---

<sup>1</sup>The term "class" is interchangeable with the term "concept" in the interpretation of ontology.

<sup>2</sup>The alternative used term is a "relationship."

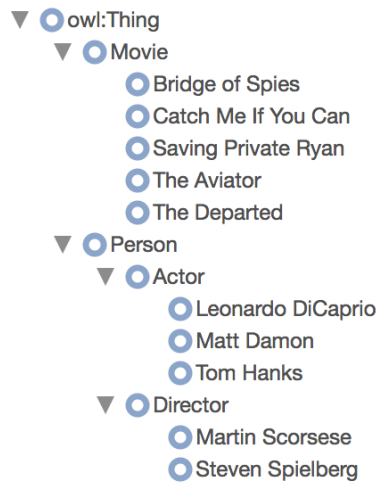


Figure 3.1: Movie class hierarchy.

In Figure 3.1 we can see the hierarchical structure of defined classes within the specific domain focused on movies. There is a root class *owl:Thing* which is the most basic class in OWL world<sup>3</sup>. The root class contains two classes *Movie* and *Person*, where *Person* has two sub-classes *Actor* and *Director*. We can consider other classes as instances of their super-classes.

Figure 3.2 presents the movie ontology graph based on the hierarchical structure described above. On the graph, we can notice a number of edges which represents various relations between the classes. Yellow arrows denote that a class belongs to its super-class, whereas blue arrows indicate named relations between classes.

Both examples, the movie class hierarchy and the movie ontology graph, were created using an open-source collaborative ontology development environment, namely WebProtege<sup>4</sup>.

<sup>3</sup>How Welty et al. (2004) explains, every individual in the OWL world is a member of the class *owl:Thing*. Thus each user-defined class is implicitly a subclass of *owl:Thing*.

<sup>4</sup><https://webprotege.stanford.edu/>

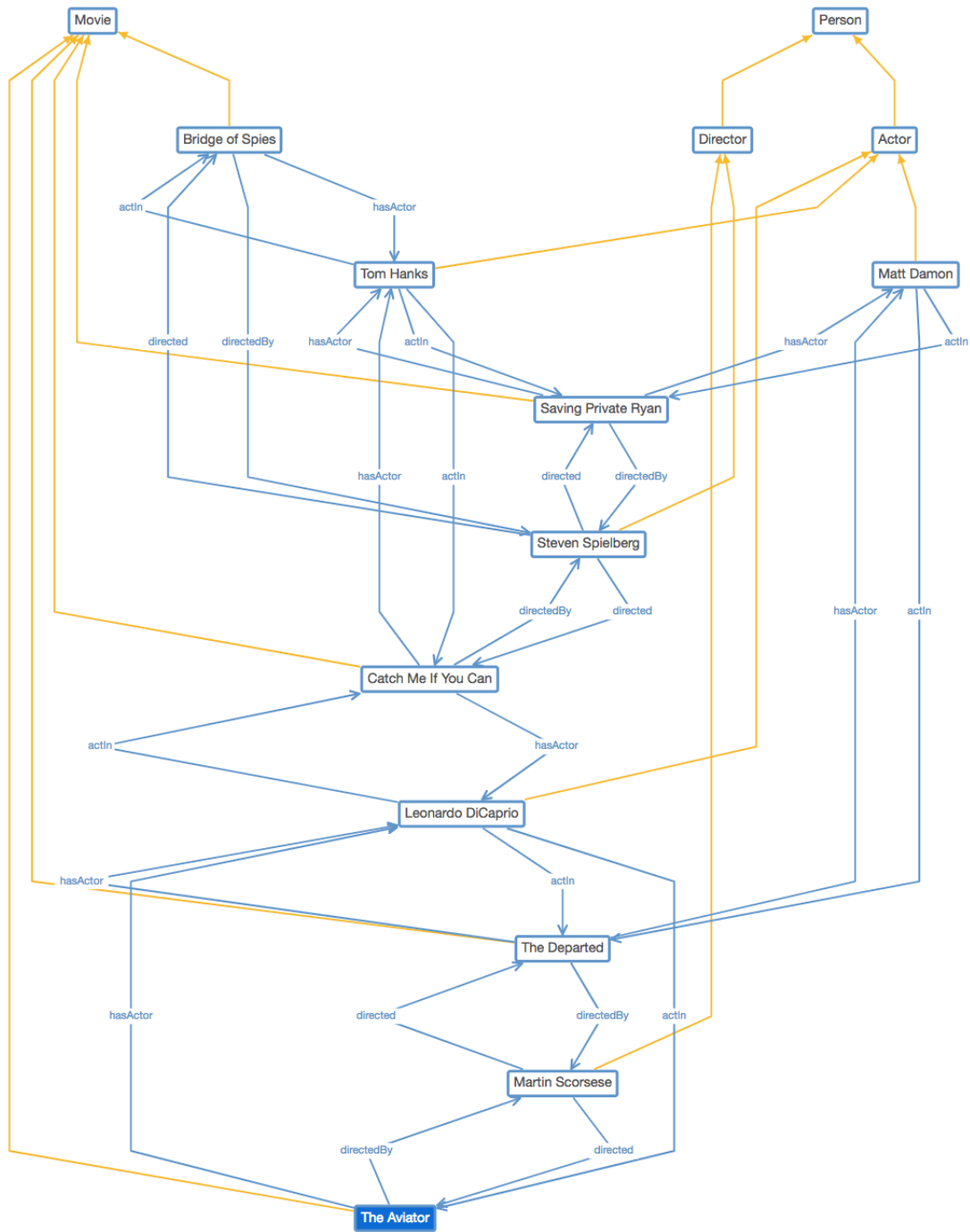


Figure 3.2: Movie ontology graph.

## 3.2 Schema.org

We could define Schema.org<sup>5</sup> as a classification scheme which is used to define entities. How Schema.org states, its existing schema is a set of 'types'<sup>6</sup>, each associated with a set of properties, and the types are arranged in a hierarchy. Selected examples of such types are Person<sup>7</sup>, Movie<sup>8</sup>, Place<sup>9</sup>. Schema.org covers several domains.

Each type has properties that can be used to describe a concrete item in more detail. For instance, the type Movie, which falls under the category CreativeWork<sup>10</sup>, can have the properties "about"<sup>11</sup>, "actor"<sup>12</sup>, "director"<sup>13</sup>, etc.

## 3.3 Knowledge Graph

Knowledge graph is a knowledge base in the shape of a graph created using a classification scheme. Such a knowledge graph provides detailed information about a specific domain or a cluster of domains using structured data and relations among them.

Figure 3.3 is a classic example of the knowledge graph. The nodes represent entities, such as Leonardo da Vinci, Mona Lisa or The Louvre and the edges describe the relationships among them.

The movie ontology graph shown in Figure 3.2 can also be considered to be a knowledge graph.

## 3.4 RDF Triple

The Resource Description Framework (RDF) is a general method for data modeling and a conceptual description of information. The standard was developed and agreed upon by W3C<sup>15</sup>. The framework can be used to structure, to publish and interlink data on the Web. RDF standard enables knowledge to be represented in a machine-readable way.

---

<sup>5</sup><https://schema.org/>

<sup>6</sup>This means 'classes' in the interpretation of ontology.

<sup>7</sup><http://schema.org/Person>

<sup>8</sup><http://schema.org/Movie>

<sup>9</sup><http://schema.org/Place>

<sup>10</sup><http://schema.org/CreativeWork>

<sup>11</sup><http://schema.org/about>

<sup>12</sup><http://schema.org/actor>

<sup>13</sup><http://schema.org/director>

<sup>14</sup>Taken from <https://aws.amazon.com/neptune/>

<sup>15</sup><https://www.w3.org/>

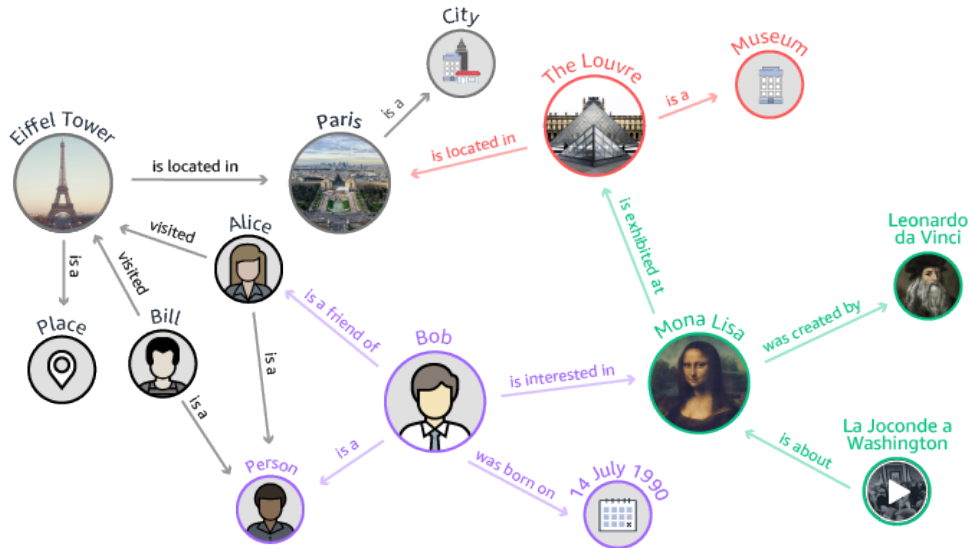


Figure 3.3: Amazon Neptune Knowledge Graph based on different widespread ontologies.<sup>14</sup>

We could define an RDF triple<sup>16</sup> as the atomic data entity in the RDF data model, specifically, as a set of three entities that codifies a statement about semantic data in the form

$$(Subject, Predicate, Object), \quad (3.1)$$

where the subject is an IRI or a blank node; the predicate is an IRI; the object is an IRI, a literal or a blank node shown in Figure 3.4. IRIs, literals and blank nodes are known as RDF terms. An RDF triple expresses a relationship between two resources<sup>17</sup>. Manola et al. (2014) explains, the subject and the object represent the two resources being related; the predicate represents the nature of their relationship. The relationship is phrased in a directional way, i.e., from subject to object, and is called a property in RDF.

RDF triples can be expressed in text format by triple languages, e.g., Turtle or N-Triples. For instance, the sentence "James Cameron directed the movie Titanic"

<sup>16</sup>An RDF triple is also known as a semantic triple or an RDF statement.

<sup>17</sup>Manola et al. (2014) states that anything can be a resource, including physical things, documents, abstract concepts, numbers, and strings.

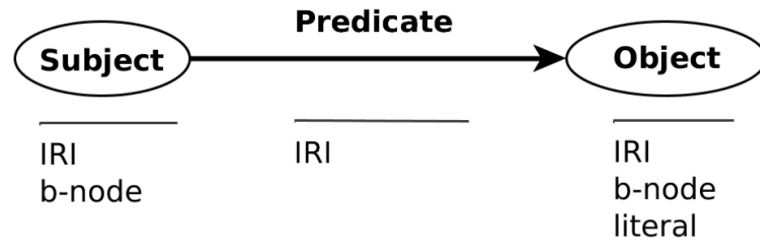


Figure 3.4: RDF graph.<sup>18</sup>

can be represented as RDF triple with DBpedia<sup>19</sup> URIs in Turtle format, i.e.

```
< http://dbpedia.org/page/James_Cameron >
  < http://dbpedia.org/ontology/director >
  < http://dbpedia.org/page/Titanic_(1997_film) > .
```

A set of RDF triples creates an RDF graph. The knowledge graph shown in Figure 3.3 can be transformed into an RDF graph if each node and each edge are represented by an IRI.

### 3.4.1 RDF Store

RDF store or Triplestore<sup>20</sup> is a purpose-build database management system for storage and retrieval of any type of data expressed in RDF. Triplestore can be seen as a subclass of graph databases<sup>21</sup>. In addition, RDF store supports SPARQL, which is a query language to retrieve RDF data from triplestores and will be discussed in Section 3.5.

As Modoni et al. (2014) mentions, a triple store architecture is based on the NoSQL paradigm because it is more suitable for exploiting the graph form of RDF data. A graph model based database, which isn't depending on a rigid schema, fits more properly to the flexible structure of RDF data.

<sup>18</sup>Taken from [https://cw.fel.cvut.cz/wiki/\\_media/courses/osw/lecture-02rdfs-s.pdf](https://cw.fel.cvut.cz/wiki/_media/courses/osw/lecture-02rdfs-s.pdf)

<sup>19</sup><https://wiki.dbpedia.org/>

<sup>20</sup>The interchangeable terms

<sup>21</sup>The databases that use graph structures (i.e. nodes and edges, where an edge is interpreted as a relation between two nodes) to represent and store data.

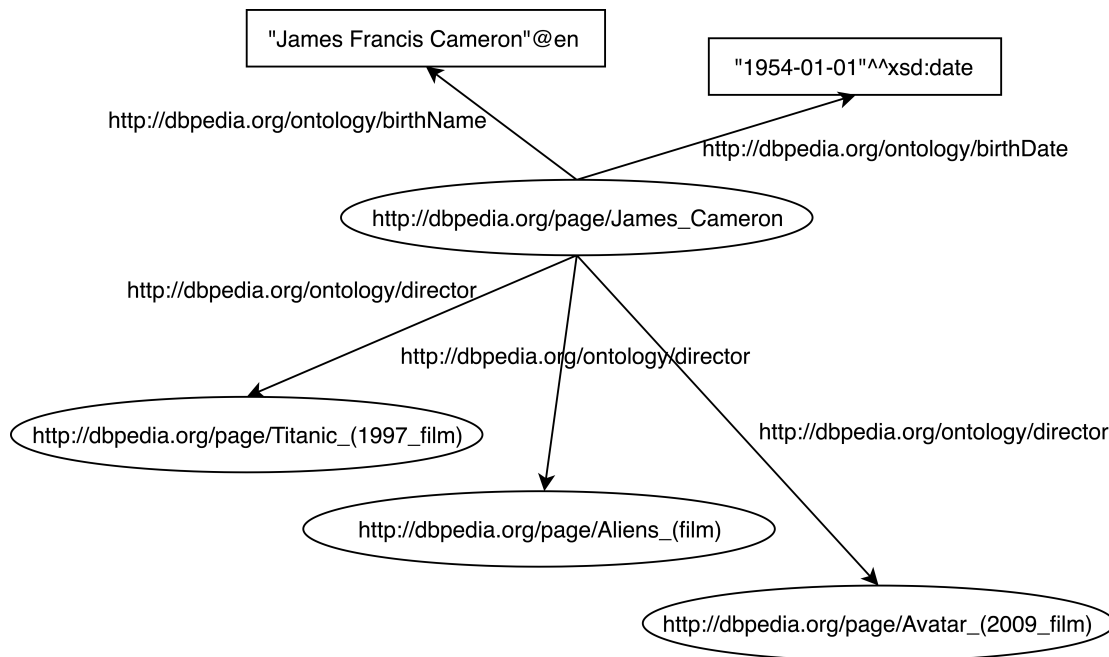


Figure 3.5: Movie RDF graph based on knowledge taken from DBpedia.

The way how data is stored in the RDF database is the following. All triples are added into a single large table with three columns that correspond to subject, predicate and object. Indexes are then created for each of the columns.

The widespread examples of RDF databases are DBpedia and Wikidata databases. DBpedia extracts facts from Wikipedia<sup>22</sup> articles and publishes them as RDF triples. Wikidata is an open knowledge base for storage of structured data. How Vrandečić and Krötzsch (2014) state, the database was created to manage data from Wikipedia. Both databases support SPARQL and provide SPARQL endpoint<sup>23</sup>.

### 3.5 SPARQL

SPARQL is a query language which is able to retrieve and manipulate data stored in RDF format. SPARQL queries are executed against RDF datasets, consisting of RDF graphs. For distant retrieving data from RDF stores, a SPARQL endpoint is used. It accepts queries and returns results via HTTP.

<sup>22</sup><https://www.wikipedia.org/>

<sup>23</sup>The term is discussed in Section 3.5.



Assume we are given an RDF graph<sup>24</sup> based on knowledge taken from DBpedia

```

< http://dbpedia.org/page/James_Cameron >
  < http://dbpedia.org/ontology/director >
    < http://dbpedia.org/page/Titanic_(1997_film) >;
  < http://dbpedia.org/ontology/director >
    < http://dbpedia.org/page/Aliens_(film) >;
  < http://dbpedia.org/ontology/director >
    < http://dbpedia.org/page/Avatar_(2009_film) >;
  < http://dbpedia.org/ontology/birthName >
    "James Francis Cameron"@en;
  < http://dbpedia.org/ontology/birthDate >
    "1954-8-16" .

```

The graph is also shown in Figure 3.5. We want to retrieve the information from the given RDF graph such as

*"What is the date of birth of James Cameron?"*. (3.2)

In this case, we would connect to DBpedia SPARQL endpoint<sup>25</sup> and execute a specific SPARQL query to obtain desired data. Such SPARQL query can look like

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?person ?dateOfBirth
WHERE {
  ?person rdf:type <http://dbpedia.org/ontology/Person >.
  ?person dbo:birthDate ?dateOfBirth .
  ?person dbp:name ?name .
  ?name bif:contains "'James Cameron'" .
  FILTER(REGEX(STR(?dateOfBirth),"[0-9]{4}-[0-9]{2}-[0-9]{2}"))
}.

```

The SPARQL query syntax, used in the previous example, is slightly similar to SQL query syntax, but certainly has nuance. So, let's take a look at the query structure, which has three main blocks. The first block is so-called *prefix declarations* which serve for abbreviating URIs. In our example, given prefix means that

<sup>24</sup>The used syntax is in Turtle format.

<sup>25</sup><https://dbpedia.org/sparql>

for the URI we use the abbreviation "rdf". The second block defines *result clause*, i.e., identifies what information to return from the query. Generally, there are four basic query types, i.e., SELECT, ASK, CONSTRUCT and DESCRIBE, so as shown above, we use SELECT which returns a binding table with results. The DISTINCT removes duplicates from returned results. Then, we have two variables starting with the character "?", the variables define which results will be returned. In our case, we are querying for values of the variable ?person and ?dateOfBirth. Finally, the third block is called *query pattern*, which specifies what to query for in the chosen database. Here, we define variables using their relations as triples. Specifically, in the first triple the variable ?person is defined as a type Person. The second triple indicates that the variable ?person has a date of birth, in which we are interested. And, the third and fourth triples say "We are interested in values of the variable person which has the specific name which contains the string "James Cameron". Ultimately, the last row in the query pattern is an option of filtering results, i.e., we are searching for a value of the variable ?dateOfBirth which would contain the date in the specific format.

After executing the query defined in (3.2) over the RDF graph, depicted in Figure 3.5, we would obtain the result table, shown in Table 3.1.

<b>person</b>	<b>dateOfBirth</b>
<a href="http://dbpedia.org/resource/James_Cameron">http://dbpedia.org/resource/James_Cameron</a>	1954-08-16

Table 3.1: SPARQL query results returned for (3.2).

We would like to give one more example of querying over the given RDF graph defined above and also shown in Figure 3.5. We can formulate the query as

*"We are interested in all movies directed by James Cameron".* (3.3)

The appropriate SPARQL query will look like

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?movie ?movieLabel
WHERE {
    ?movie rdf:type <http://dbpedia.org/ontology/Film>.
    ?movie dbo:director ?director .
    ?movie rdfs:label ?movieLabel .
    ?director rdf:type <http://dbpedia.org/ontology/Person>.

```

```

?director dbp:name ?directorLabel .
?directorLabel bif:contains "James Cameron" .
FILTER(lang(?movieLabel) = "en") .
}.
```

The results for the executed SPARQL query on the given RDF graph, shown in Figure 3.5, can be seen in Table 3.2.

<b>movie</b>	<b>movieLabel</b>
<a href="http://dbpedia.org/resource/Titanic_(1997_film)">http://dbpedia.org/resource/Titanic_(1997_film)</a>	"Titanic (1997 film)"@en
<a href="http://dbpedia.org/resource/Aliens_(film)">http://dbpedia.org/resource/Aliens_(film)</a>	"Aliens (film)"@en
<a href="http://dbpedia.org/resource/Avatar_(2009_film)">http://dbpedia.org/resource/Avatar_(2009_film)</a>	"Avatar (2009 film)"@en

Table 3.2: SPARQL query results returned for (3.3).

# Chapter 4

## Natural Language Processing

Natural language processing is a sub-field of AI which is focused on enabling computers to understand natural human language. The existing NLP techniques make efforts to bring computers closer to the understanding of human language.

In this chapter, we overview the issues that are faced in NLP. We describe word-sense disambiguation, part-of-speech recognition, named-entity recognition, anaphora resolution and the need of maintaining context of a conversation. We end the chapter with a look at natural language generation.

### 4.1 Natural Language Understanding

Natural language understanding is a subtopic of NLP which is focused on machine reading comprehension, i.e., deals with text processing, e.g., data extraction, and understanding of data meaning. NLU is considered to be an AI-hard problem.

The main tasks of NLU are information extraction are intent recognition and named entity recognition.

#### 4.1.1 Intent Classification

As stated in e.g. Hutson (2017), intent is the underlying meaning of an utterance. For example, in the question

*”Compare cell phones from Apple with those from Samsung.”*

the intent is that the user wants to know a comparison between some elements. It is not easy to classify intents in general due to their comprehensive expressive power.

To follow with other examples, observe that in the sentences

*"Make me a coffe please."*

*"Who is the director of Titanic?"*

*"Turn the heating on."*

*"How is glass manufactured?"*

*"What do you think about Plato's Cave?"*

the structure of the intent is completely different.

For our purposes, we can restrict ourselves to closed-domain systems. As an example, Yan et al. (2017) builds a task-oriented chatbot for online shopping. Their intents come from a pre-defined set – it contains e.g. asking for recommendation, asking for comparison, asking opinions or querying attributes of products. To form a whole representation of the understanding of a user query, they use a triple

$$(I, C, A), \tag{4.1}$$

where  $I$  is the user intent,  $C$  are the categories of products that were mentioned in the query and  $A$  is a list of mentioned attributes and their values. Yan et al. (2017) then approaches the task of classifying these values by a convolutional neural network.

Bhattacharyya and Hazarika (2016) takes a more theoretical approach to intents and tries to classify a general human intent by a hierarchical finite state machine.

### 4.1.2 Named-entity Recognition

Named-entity recognition (NER) is an important sub-task of NLP, which can be used to extract structured data from unstructured text. In detail, it aims to tag each word that defines a specific entity by a label from a pre-defined set.

For instance, we might want to label personal names and names of places in text. Then, for the input string

*"Angela Merkel gave a speech at the conference in Munich."*

the tagged entities should be "Angela Merkel" as a person and "Munich" as a place.

According to Jurafsky and Martin (2014), the typical approach for NER is to extract the features of the individual words occurring in the text – such features could be for instance the part-of-speech tag of the word, its embedding, presence of a particular suffix or a prefix in the word or the shape of the word. Word embedding stands for a vector representation of a word that satisfies useful properties. We will talk about them in detail in Section 5.3. By shape, it is meant which characters in

the word are upper-case and which are lower-case, the occurrence of numbers and punctuation. For example, the words "Munich" and "Angela" share the same shape, because they both start with an upper-case letter and continue with 5 lower-case letters. On the other hand, the words "John", "Ing." or "bear" have all different shapes (the difference between the first two is in the fact that dot is not a lower-case letter, but a punctuation mark).

There could also be additional features based on lists of possible named entities – for identifying names of persons, we can use e.g., lists of used names of persons. For names of places, such list is called a gazetteer, which is a term that is also used as a general name for such lists of named entities (gazetteer has two possible meanings, in the following text we will use the second one). If we had multiple gazetteers, we could add a binary feature for each of them that would mark whether a particular word occurs in the gazetteer or not. Observe that a single name may be in multiple lists – for example, Washington is a personal name, a state, and a city. Mikheev et al. (1999) discusses and shows that large lists of names may not be necessary for particular types of texts and that a short list with most frequent names may be sufficient.

Because the given text is viewed as a sequence of words, it is also beneficial to incorporate some of the features of the neighboring words into the features of the current word. This means that when we are constructing the features of the word "Merkel" from the sentence given in the example above, we also may add the information concerning the previous word "Angela" and the following word "gave". Of course, we can also use a wider neighborhood (i.e. consider also the word after the following word etc.).

As mentioned in Jurafsky and Martin (2014), currently used academical state-of-the-art approaches for NER are based on an RNN with LSTM cells and/or utilize conditional random fields, whereas in commercial applications, one can resort to more simple rule-based methods that rely more on the available gazetteers.

## Stanford NER

The Stanford Natural Language Processing Group<sup>1</sup> deals with various NLP tasks and provides high-performing implementations that solve them. One of them is also their NER tagger, which is based on a CRF classifier. We will briefly discuss the motivation for using CRF and its usage for NER.

If we are given two sets of random variables  $X$  and  $Y$ , where  $X$  are the observed variables and  $Y$  are the unknown (hidden) variables, we might be interested in modeling the probability distribution  $p(X, Y)$ , which can be done in general for example by a Markov random field. However, we would then want to find for each

---

<sup>1</sup>[nlp.stanford.edu](http://nlp.stanford.edu)

realization  $x$  of variables in  $X$  the realization  $y$  of variables in  $Y$  such that the value of  $p(Y = y|X = x)$  is maximal. A novel class of graphical models, called Conditional Random Fields, which were presented in Lafferty et al. (2001) are focused on modeling the conditional probability distribution  $p(Y|X)$  directly instead of modeling the joint distribution.

To be used in NER, the hidden variables  $Y$  correspond to the tags that we want to assign to the words and the known variables  $X$  correspond to the words to be labeled, respectively their features. Then, for given words, a CRF can be used to find the most probable tags. A more detailed introduction to CRFs can be found for instance in Sutton et al. (2012) or Sutton and McCallum (2006).

### 4.1.3 Anaphora Resolution

When holding a conversation, it is common in human communication to remember its context. For instance, when a person says

*"Yesterday I watched the new movie Arrival. Have you seen **it** already?"*

it is not necessary to repeat the name of the movie in the second sentence, but only refer to it using a pronoun "it", as it is assumed to be deduced from the context of the previous sentence. Such coreferences are called anaphoras and the process in which the references are replaced by the original terms is called anaphora resolution. This is an important and difficult task in NLP, because the computer itself can not reason about the coreferences. As mentioned in Nemcik (2006), this task itself is often considered to be AI-complete.

However, anaphora resolution on a single utterance may not be always sufficient – assume that the dialog started with the previously shown sentence and continues as follows:

*"No, not yet."*

*"You should, **it** was quite good."*

then, the coreference is linked through two exchanges, but one can easily imagine that it would go through even more. That is why chatbots need to maintain the context of the conversation even after multiple exchanges in order to correctly understand given sentences.

## 4.2 Natural Language Generation

Natural Language Generation is a crucial part of each chatbot, since it is responsible for producing the actual response of the bot to the user. In NLG in general, multiple tasks must be addressed to produce a text that is readable for humans. Reiter and Dale (2000) lists these tasks as follows:

- **Content determination** decides which information should be included in the response and which should be omitted.
- **Document structuring** determines in which order the information should be present.
- **Lexicalisation** chooses the particular words that will be used in the text and deals for instance with the choice among synonyms.
- **Referring expression generation** decides whether a word will be mentioned by its full name, by description, or just by using a pronoun. For instance, we would like to produce the text

”Leonardo DiCaprio starred in the movie Titanic. It was made in 1997.”

rather than

”Leonardo DiCaprio starred in the movie Titanic. The movie Titanic was made in 1997.”

- **Aggregation** makes sure that if there were some unnecessary repetitions in the text, they should be put together. For example, instead of

”Leonardo DiCaprio starred in the movie Titanic. Kate Winslet starred in the movie Titanic.”

we should rather formulate it as

”Leonardo DiCaprio and Kate Winslet starred in the movie Titanic.”

- **Linguistic realisation** transforms the results of the above tasks into an actual text by for example putting the words into the correct form according to the grammar of the language.
- **Structure realisation** prepares the already created text into the correct form based on the system in which it will be presented – it may for example add tags or annotations to distinguish individual paragraphs of a longer text.

Currently, the most successful approaches for NLG in conversational chatbots, such as Chen et al. or Pichl et al., are based on text templates that are prepared in advance. Then, during the conversation, a corresponding template is chosen as a response, into which additional information may be filled using slots. Template-based NLG may in fact correspond to various different approaches<sup>2</sup>, but in its simplest form it corresponds to having a template sentence, in which some entities may be missing, for example the template

---

<sup>2</sup>It also may not be possible to distinguish it sharply from other approaches (different types of text templates), as discussed in Deemter et al. (2005)



"<actor> starred in <movie>."

produces the sentence

"Leonardo DiCaprio starred in Titanic."

by the assignment <actor>  $\rightarrow$  "Leonardo DiCaprio", <movie>  $\rightarrow$  "Titanic".

Except for template-based approaches, there are also methods based on the formalization of the grammar of a natural language or statistical machine-learning methods, whose representative is for example Wen et al. (2015) that uses an RNN to generate natural language output.

Even though template-based NLG may seem to be overly restrictive due to the large extent of required human labor (templates must be written manually by the programmer), Deemter et al. (2005) compares it to the other approaches and comes to the conclusion that templates should not be viewed as inferior.

# Chapter 5

## Machine Learning Algorithms for NLP

In this chapter, we are first of all going to give a general introduction into fitting models to training data and estimating the error of models. Then, we describe feed-forward neural networks and recurrent neural networks. After that, we again focus on the usage of NNs in NLP, namely for the creation of word embeddings. We conclude the chapter by introducing the Multi-armed bandit problem, which exemplifies the exploration-exploitation dilemma that we will also face in our implementation.

### 5.1 Training of Models and Evaluation

We would like to describe the general principles of statistical machine learning, i.e., a model fitting on data for classification problem, specifically focusing on supervised learning. Then, the common methods of its evaluation will be discussed. The section is based on material from Mohri et al. (2018).

More formally, in a supervised classification problem, we use the following concepts:

- $X$  is a set of input observations/features
- $Y$  is a finite set of hidden states<sup>1</sup>
- $T$  is a set of data

$$T = \{(x_i, y_i) \in X \times Y \mid i \in \{1, \dots, m\}\} \quad (5.1)$$

i.e. pairs of values, where  $x_i$  is an observation from  $X$  and  $y_i$  is its corresponding true hidden state from  $Y$

---

<sup>1</sup>The hidden values or labels, respectively the values to be predicted.

- $h_\theta : X \rightarrow Y$  is a prediction strategy<sup>2</sup> depending on parameters  $\theta$  which should be learned
- $l : Y \times Y \rightarrow \mathbb{R}$  is a loss function which penalizes wrong predictions, i.e.  $l(y, h_\theta(x))$  is the loss for predicting  $y' = h_\theta(x)$  when  $y$  is the true state

In addition, we could assume that  $X$  and  $Y$  are random variables, their values can be obtained by, e.g., running an experiment or observing an environment. So, there could be a probability distribution  $p$  that would assign each pair  $(x, y) \in X \times Y$  a probability of observing it as  $p(x, y)$ .

The goal of a classification problem is to find a prediction strategy with the minimal expected risk, where the expected risk can be calculated as

$$R(h_\theta) = \sum_{(x,y) \in X \times Y} l(y, h_\theta(x)) p(x, y) = \mathbb{E}_{p(x,y)} [l(y, h_\theta(x))]. \quad (5.2)$$

However, the probability distribution  $p$  is usually unknown, so we consider using testing data

$$T_{\text{test}} = \{(x_i, y_i) \in X \times Y \mid i \in \{1, \dots, n\}\}. \quad (5.3)$$

Now, using (5.3) the empirical risk can be calculated as

$$R_{T_{\text{test}}}(h_\theta) = \frac{1}{|T_{\text{test}}|} \sum_{(x,y) \in T_{\text{test}}} l(y, h_\theta(x)) \quad (5.4)$$

which can be viewed as an approximation for  $R(h_\theta)$ .

In the case, when  $l(y, h_\theta(x))$  is so-called a 0-1 loss function, which can be defined as

$$l(y, h_\theta(x)) = \begin{cases} 0 & \text{if } y = h_\theta(x) \\ 1 & \text{if } y \neq h_\theta(x) \end{cases}, \quad (5.5)$$

the  $R(h_\theta)$  is called the *generalization error* or the *expected error* of a prediction strategy, and  $R_{T_{\text{test}}}(h_\theta)$  is called the *empirical error* of a prediction strategy, because these quantities correspond to the probability of making wrong predictions.

Theoretically, we would like to find a set of classification models for  $h_\theta$  parameterized by  $\theta$ , so the value of  $R(h_\theta)$  would be minimal. Practically, it is usually not possible to calculate the expected risk, because  $p(x, y)$  is unknown, thus we need to estimate it. The commonly used methods for estimation will be described in the next section.

---

<sup>2</sup>Also known as an inference rule

### 5.1.1 Training, Validation, and Testing Datasets

The empirical risk should be calculated using testing data as was described above. In the case, when the empirical risk is calculated using all available non-separated data, it can lead to model overfitting. Overfitting means that a model is performing well on training data, but classifies poorly on unobserved data. We could say that the risk of overfitting is higher when more flexible models are used.

To avoid overfitting, it is necessary to divide the available non-separated data  $T$  into three pairwise disjoint datasets  $T_{\text{train}}, T_{\text{val}}, T_{\text{test}}$ . The  $T_{\text{train}}$ <sup>3</sup> dataset would be used for training, i.e. for the estimation of parameters  $\theta$  of a model. The  $T_{\text{val}}$ <sup>4</sup> dataset would be used to calculate the empirical risk and to find the hyper-parameters<sup>5</sup> of a model. The  $T_{\text{test}}$ <sup>6</sup> dataset, whose data should be unobserved during training, would be used for evaluating a finally chosen model, i.e., the best performing model on  $T_{\text{val}}$ , to estimate its performance.

### 5.1.2 Cross-validation

Cross-validation is used in the situation when  $T_{\text{train}}$  and  $T_{\text{val}}$  datasets haven't enough data to train a model properly or to make a precise comparison among hyper-parameters. The most widespread type of cross-validation is a so-called  $k$ -fold cross-validation.

In  $k$ -fold cross-validation, the available data is randomly divided into  $k + 1$  pairwise disjoint equal sized subsets

$$T_1, \dots, T_k, T_{\text{test}}. \quad (5.6)$$

The purpose of  $T_{\text{test}}$  subset is the same, i.e. for evaluation of a finally chosen model, as was described above.

The  $k$  subsets are used for training and validation. We train  $k$  models  $h_{\theta_1}, \dots, h_{\theta_k}$  where the model  $h_{\theta_i}$ ,  $i \in \{1, \dots, k\}$  is trained using data from the set

$$\bigcup_{j \in \{1, \dots, k\} - \{i\}} T_j, \quad (5.7)$$

i.e. data from all sets  $T_j$  except for the set  $T_i$ . Then, each of these models is evaluated exactly on the set  $T_i$ .

---

<sup>3</sup>Training dataset

<sup>4</sup>Validation dataset

<sup>5</sup>E.g., the choice of the kernel and its dimension in SVM or the number of hidden layers in an NN.

<sup>6</sup>Testing dataset or so-called holdout data

The overall cross-validation error is thus calculated as

$$\frac{1}{k} \sum_{i=1}^k R_{T_i}(h_{\theta_i}) = \frac{1}{k} \sum_{i=1}^k \frac{1}{|T_i|} \sum_{(x,y) \in T_i} l(y, h_{\theta_i}(x)). \quad (5.8)$$

This criterion can be used for comparison among the hyper-parameters of the model.

## 5.2 Artificial Neural Networks

Artificial neural networks are a hugely useful and extraordinarily successful tool of machine learning in general. In this section, we are going to briefly describe them – we will start with feedforward neural networks, describe their layers and parameter learning while avoiding overfitting. Secondly, we will explain the differences in recurrent neural networks and in detail look on LSTM cells that are often utilized in NLP. The section is based on material from Goodfellow et al. (2016).

### 5.2.1 Feedforward Neural Network

A feedforward neural network can be viewed as a vector function, which means that it accepts multiple values on its input and also produces multiple outputs. More formally, it is a function  $f_{\theta} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  that has parameters  $\theta$ .

Neural networks are usually composed of layers, which can be viewed as simple intermediate vector functions

$$f_{\theta_j}^j : \mathbb{R}^{d_j} \rightarrow \mathbb{R}^{d_{j+1}}, \quad (5.9)$$

so that

$$f_{\theta} = f_{\theta_1}^1 \circ f_{\theta_2}^2 \circ \dots \circ f_{\theta_h}^h, \quad (5.10)$$

where  $d_1 = n$ ,  $d_{h+1} = m$  and  $\theta$  contains all the parameters of the intermediate functions. The first layer is usually called the input layer, the last layer is the output layer and the ones between them are called hidden layers.

There is a multitude of possible layers that can be used, but simple feedforward neural networks are usually composed of fully-connected layers followed by a non-linear activation function, which is again followed by a fully-connected layer, etc. – the kinds of layers alternate until the last layer is reached. If we want the NN to perform classification, softmax is commonly used as the last layer.

The fully-connected layer  $j$  corresponds to a function in the form

$$f_{A,b}^j(x') = Ax' + b, \quad (5.11)$$

where  $A$  is a real matrix of size  $d_{j+1} \times d_j$  and  $b$  is a bias vector of length  $d_{j+1}$ . The parameters of such layer are the values in the matrix and in the bias vector.

The layers corresponding to activation functions have the same amount of inputs as outputs, because they correspond to element-wise application of a non-linear function. More formally, if we consider a layer  $j$  to be such, it holds that  $d_j = d_{j+1}$  and

$$f_{\theta}^j(x') = \begin{pmatrix} g(x'_1) \\ g(x'_2) \\ \vdots \\ g(x'_{d_j}) \end{pmatrix}, \quad (5.12)$$

where  $g : \mathbb{R} \rightarrow \mathbb{R}$  is an activation function that usually does not have any parameters, so this activation layer does not have any by itself.

It is important that the activation function is non-linear. Otherwise, it could be modeled by the directly preceding fully-connected layer and become useless. As stated for example in Ramachandran et al. (2017), the currently most-successful and widely-used activation function is ReLU, i.e.

$$g(x') = \begin{cases} x' & \text{if } x' \geq 0 \\ 0 & \text{if } x' < 0. \end{cases} \quad (5.13)$$

However, there might appear some issues with this function as it has zero derivative for all  $x' < 0$  and its derivative is undefined at  $x' = 0$ . Due to the zero derivative, the backpropagation algorithm, that will be introduced later, may fail to update the parameters that are "behind" this element, which might be a problem in some architectures. The issue with zero derivative is solved for example by LeakyReLU, which is defined as

$$g(x') = \begin{cases} x' & \text{if } x' \geq 0 \\ \alpha x' & \text{if } x' < 0 \end{cases} \quad (5.14)$$

for some small positive constant  $\alpha$ . There are also other modifications of ReLU, such as ELU, which was presented in Clevert et al. (2015), SELU, proposed in Klambauer et al. (2017), and others.

Another activation functions are for instance hyperbolic tangent, defined as

$$\tanh(x') = \frac{e^{x'} - e^{-x'}}{e^{x'} + e^{-x'}}, \quad (5.15)$$

or logistic sigmoid function, defined as

$$g(x') = \frac{1}{1 + e^{-x'}}. \quad (5.16)$$

In case that we would like to create a neural network for classification, the last

layer in the network is so-called softmax, which is calculated as

$$f_{\emptyset}^h(x') = \begin{pmatrix} r(1, x') \\ r(2, x') \\ \vdots \\ r(d_h, x') \end{pmatrix}, \quad (5.17)$$

where

$$r(q, x') = \frac{e^{x'_q}}{\sum_{j=1}^{d_h} e^{x'_j}}. \quad (5.18)$$

This layer does not have any parameters and is used for classification into  $m$  classes so that we identify each of the  $m$  (resp.  $d_h = m$ ) classes with a single value of the vector on the output and then, when an input  $x$  is put into the network, we classify it as the class whose corresponding value on the output of the last layer (i.e. the softmax layer) has the highest value.

## Training of Neural Networks

In the previous section, the structure of feedforward NN was described in general, now we would like to show how to train such model. Assume that we are given training data

$$T = \{(x_i, y_i) \mid i \in \{1, \dots, l\}\}, \quad (5.19)$$

where  $x_i \in \mathbb{R}^n$ ,  $y_i \in \mathbb{R}^m$ , we would like to find the parameters  $\theta$  of the neural network so that it will hold  $f_{\theta}(x_i) \approx y_i$  for all  $i \in \{1, \dots, l\}$ .

To do so, we need to create an additional layer to the neural network, which is called the loss layer. The choice of loss layer depends on the task that we want to model. In the case of regression, the sum of squared errors is used as the loss layer. This results in a function

$$f'_{\theta}(T) = \sum_{(x,y) \in T} \sum_{j=1}^m (f_{\theta}(x)_j - y_j)^2, \quad (5.20)$$

where  $f_{\theta}(x)_j$  is the  $j$ -th element of the predicted vector for input  $x$  and  $y$  is the correct (also called target) vector.

On the other hand, if we perform classification into multiple (let us say  $m$ ) classes, the vectors  $y$  in the training set are usually encoded as one-hot vectors that represent an index of the correct class. In such case, one should use categorical cross entropy as the loss layer, which yields

$$f'_{\theta}(T) = - \sum_{(x,y) \in T} \sum_{j=1}^m y_j \log(f_{\theta}(x)_j), \quad (5.21)$$

where  $f_\theta(x)_j$  is the  $j$ -th element of the output vector that was created by the softmax layer.

The learning of the network then corresponds to minimizing the value of  $f'_\theta(T)$  concerning the parameters  $\theta$ . For this task, a method called backpropagation is used. Because the function  $f$  is composed of simple functions, it is easy to calculate its gradient w.r.t. the parameters  $\theta$  by chain rule and we can, therefore, do the same for  $f'$ . We could then apply a gradient descent method to find a minimum. However, the evaluation for the whole dataset may be time-consuming, so only a small portion (called batch) is used to calculate the value of the gradient. This approach is called stochastic gradient descent and is based on iterating over batches of training data – for each batch, the output of the network and gradient are calculated and then, the parameters of the network are updated.

## Techniques for Improving Generalization

Because neural networks form flexible models, it is necessary to avoid overfitting, which is the case that the model predicts well on the training data set, but has a large error on data that were not used in training. To avoid this, one can apply various measures to prevent it and improve generalization; such measures are:

- Using a **validation data set** that is periodically evaluated while training the neural network and once the error on the validation data set starts to increase, it is a sign of overfitting on the training dataset, so the training is stopped. This procedure is also called **early stopping**.
- Introducing **dropout** that "turns off" some neurons in the neural net with a given probability. This formally corresponds to setting the value of some components on the output in the intermediate layers to zero. This procedure forbids the neural network from, e.g., relying on a single input feature.
- Adding a **regularization** term to the loss function – this formally corresponds to minimizing

$$f'_\theta(T) + \alpha \sum_{i=1}^{|\theta|} \theta_i^2 \quad (5.22)$$

in case of  $L^2$  regularization, which is more frequently used, or to minimization of

$$f'_\theta(T) + \alpha \sum_{i=1}^{|\theta|} |\theta_i| \quad (5.23)$$

in case of  $L^1$  regularization. In both cases,  $\alpha > 0$  is a regularization constant, and the added terms assure that the norms of the individual parameters are small, so no parameter may "outweigh" the others – again as with dropout, this forces the neural network to use all the parameters.



The last method that should be applied to obtain a well-generalizing neural network is hyper-parameter optimization. Whereas the previously mentioned methods dealt only with the values of the parameters of an NN, hyper-parameter optimization chooses the amount and the kind of parameters that the NN should have. These hyper-parameters could be categorical, such as

- used activation function and
- used optimizer

or numerical, such as

- number of hidden layers,
- the dimensionality of the vectors that the hidden layers process,
- parameters of the optimizer and
- dropout probability.

For each of the desired hyper-parameters, we should first use reasoning based on professional experience to set up a range of its possible values. For continuous hyper-parameters or integer hyper-parameters with a large range, the values should be sampled as it would not be possible or not be useful to evaluate all the values. Then, we should apply a method of choosing the best combination of them.

Such method could be for example grid search, in which we evaluate all the combinations of the values of the parameters and then pick the best-performing one. The disadvantage of this procedure is that there could be exponentially many combinations with respect to the number of hyper-parameters and that is also why we should think about sub-sampling the possible values for each hyper-parameter when using grid search. We should also remember that the evaluation for a single choice of hyper-parameters is costly because it corresponds to training of a possibly large neural network with many training data.

Another widely-used option according to Bergstra and Bengio (2012) is called manual search, in which the tried combinations are manually crafted based on the results on the previous runs on different combinations.

Bergstra and Bengio (2012) claims that random search is equally or more efficient than both grid search or manual search. In a random search, we randomly generate a number of possible configurations and then evaluate them and choose the best-performing. The results are justified by claiming that usually, only a few hyper-parameters are important for having good results, but these hyper-parameters are different for each prediction problem. Therefore, evaluating the combinations of the unimportant parameters, as grid search does, seems useless.

Bergstra et al. (2011) also offers other alternatives, namely Sequential Model-Based Global Optimization. This is a general class of algorithms that are useful in cases when the evaluation of the criterion function is expensive, which is our case because the training of a neural netwo is time-demanding. These algorithms use an approximation of the true criterion function that is estimated using the already known evaluated values for some hyper-parameter combinations. The values of the hyper-parameters for the next step are then chosen as the minimizers of the approximation, which should be computable in a reasonable time. The true criterion function is then evaluated for these parameters, the approximate function is re-fit using the newly calculated value and the whole process repeats. Finally, we can choose the parameters that resulted in the best-found criterion function value. This procedure is more formally written in Algorithm 1, where the true criterion function that should be minimized is  $g$ , its parameters are denoted by  $\phi$ , and the domain from which the parameters come is the set  $\Phi$ . The approximation function trained on sample data-pairs

$$T = \{(\phi_1, g(\phi_1)), \dots, (\phi_n, g(\phi_n))\} \quad (5.24)$$

is denoted as  $M_T$  and its value at point  $\phi$  is denoted as  $M_T(\phi)$ .

---

**Algorithm 1:** Sequential Model-Based Global Optimization of function  $g$  over the domain  $\Phi$  for  $s$  steps.

---

```

1 Function SMBO( $g, s$ ) is
2    $T \leftarrow \emptyset$ ;
3   for  $i$  in  $\{1, \dots, s\}$  do
4      $\phi_i \leftarrow \operatorname{argmin}_{\phi \in \Phi} M_T(\phi)$ ;
5     calculate  $g(\phi_i)$ ;
6      $T \leftarrow T \cup \{(\phi_i, g(\phi_i))\}$ ;
7   return  $T$ ;
```

---

## 5.2.2 Recurrent Neural Networks

The previously shown feedforward neural networks only accept inputs of a fixed length. However, this may be problematic for example in NLP or speech recognition, where the inputs may vary in length and can not be put into a feedforward NN separately because there is a dependence between the individual inputs, which would be lost by their separation.

That is why recurrent neural networks were created – unlike feedforward neural nets, they have a memory, so that we can input a sequence of vectors into such network and the dependence between the inputs will be retained. The memory in RNNs is implemented by sending a part of the output of some layer as an input to

a previous layer in the network so that its value can be re-used for the next input value.

However, it was found in, e.g., Bengio et al. (1994) that training an RNN for long sequences is difficult, so it is necessary to find special kinds of cells or layers that could be used and would avoid the vanishing gradient problem. Such elements are for example LSTM cells, which were introduced in Hochreiter and Schmidhuber (1997), or Gated Recurrent Units that were inspired by LSTMs and are surveyed for example in Dey and Salem (2017).

### Long Short-term Memory

There are multiple variants of the LSTM cells, which are listed in, e.g., Greff et al. (2015), we will describe the most popular variant with forget gate. This variant is depicted in Figure 5.1. In the figure, there are three gates (input, output and forget), block input and the cell that stores the inner state. Both the block output and the input vector are connected to all the gates and the block input. Following the notation from Greff et al. (2015), the names of elements in an LSTM cell are overviewed in Table 5.1.

Symbol	Meaning
$N \in \mathbb{N}$	number of LSTM blocks
$M \in \mathbb{N}$	size of input
$x^t \in \mathbb{R}^M$	input vector at time $t$
$z^t \in \mathbb{R}^N$	block input at time $t$
$y^t \in \mathbb{R}^N$	block output at time $t$
$i^t \in \mathbb{R}^N$	input gate at time $t$
$f^t \in \mathbb{R}^N$	forget gate at time $t$
$o^t \in \mathbb{R}^N$	output gate at time $t$
$c^t \in \mathbb{R}^N$	cell value at time $t$
$W_z, W_i, W_f, W_o \in \mathbb{R}^{N \times M}$	input weights (matrices with parameters)
$R_z, R_i, R_f, R_o \in \mathbb{R}^{N \times N}$	recurrent weights (matrices with parameters)
$b_z, b_i, b_f, b_o \in \mathbb{R}^N$	bias weights (vectors with parameters)

Table 5.1: Notation of the used symbols in equations (5.25)-(5.30)

Using these elements, we can define the processing of an LSTM cell by the following discrete dynamic equations:

$$z^t = \tanh(W_z x^t + R_z y^{t-1} + b_z) \quad (5.25)$$

$$i^t = \sigma(W_i x^t + R_i y^{t-1} + b_i) \quad (5.26)$$

$$f^t = \sigma(W_f x^t + R_f y^{t-1} + b_f) \quad (5.27)$$

$$c^t = z^t \odot i^t + c^{t-1} \odot f^t \quad (5.28)$$

$$o^t = \sigma(W_o x^t + R_o y^{t-1} + b_o) \quad (5.29)$$

$$y^t = \tanh(c^t) \odot o^t, \quad (5.30)$$

where  $\sigma$  is the logistic sigmoid function applied element-wise and  $\tanh$  is also applied element-wise. The hyperbolic tangent in (5.25) and (5.30) may also be replaced by ReLU, as stated in Dey and Salem (2017). The symbol  $\odot$  is the element-wise multiplication of two vectors, it can be also generalized to matrices, where it is called Hadamard product.

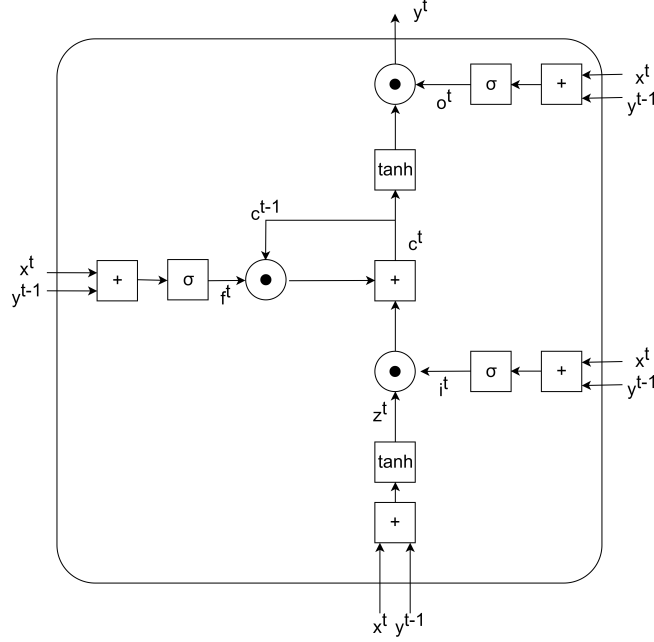


Figure 5.1: Block schema of LSTM cell.

### 5.3 Word Embedding

Many ML algorithms and deep learning architectures, such as RNN, are not capable of processing sentences in a natural language, i.e., strings or plain text in their raw form. In this case, word embedding can help.

Word embedding is a widespread technique in NLP, which maps the words from a dictionary  $W$ , where  $W \subseteq \Sigma^+$ , to a vector space in a specified dimension.

More formally, we would like to find a mapping  $f$  that assigns each word  $w \in W$  a real-valued  $d$ -dimensional vector  $f(w) \in \mathbb{R}^d$ , i.e.

$$f : W \rightarrow \mathbb{R}^d. \quad (5.31)$$

Besides, a mapping  $f$  should satisfy useful characteristics, such as words with similar meaning have a similar vector representation; synonyms can be clustered, etc. It helps to capture semantic and syntactic words relations.

### 5.3.1 Word2vec

Word2vec is a technique used to produce word embeddings. This task was already discussed by Dhillon et al. (2011), but a current state-of-the-art approach is invented by Mikolov et al. (2013a).

The approach is a group of models that use shallow neural networks which are utilized to find a mapping  $f$ .

Mikolov et al. (2013a) mentions two model architectures, both are based on sequences of words in a sentence and can be sped up by using negative sampling and hierarchical softmax.

Denote a single sentence with  $k$  words as a sequence  $\{w_i\}_{i=1}^k$  of words from  $W$ . The first model, CBOW tries to predict the middle word in the  $(2n + 1)$ -gram based on the surrounding  $2n$  words. More formally, we use the words

$$w_{i-n}, w_{i-n+1}, \dots, w_{i-1}, w_{i+1}, \dots, w_{i+n}, \quad (5.32)$$

i.e., a bag of words, to predict  $w_i$ . The second model, skip-gram performs the opposite procedure.  $w_i$  is given, and it is trained to predict the  $2n$  surrounding words.

Both model architectures return comparable results. The precision of the found mapping  $f$  depends on the amount of training data and the dimension  $d$  of the vector.

The mapping  $f$  found by the models has valuable properties, which are in detail discussed in Mikolov et al. (2013b), the vector representation maintains the semantic and syntactic words relations. For instance, if there are three words  $w, w', y \in W$ , we are searching for a fourth word  $y' \in W$  such that  $y$  is in the same relation to  $y'$  as  $w$  is to  $w'$ . Formally, we formulate it as

$$y' = \operatorname{argmin}_{x \in W} D_C(f(y) + f(w') - f(w), f(x)), \quad (5.33)$$

where  $D_C$  is the cosine distance and  $f(w') - f(w)$  can be interpreted as the "relation vector". The example results for word  $y'$  can be seen in Table 5.2 taken from Mikolov et al. (2013a).

Relationship type	$w$	$w'$	$y$	$y'$
Plural form	mouse	mice	dollar	dollars
Antonym	possible	impossible	ethical	unethical
Capital city of state	Athens	Greece	Oslo	Norway
Man-woman	brother	sister	grandson	granddaughter

Table 5.2: Examples of preserved relationships.

### 5.3.2 GloVe

GloVe is a model for distributed word representation. It includes the overall structure of word frequencies, i.e., it uses a square matrix  $X$  of size  $|W| \times |W|$ , whose elements  $X_{i,j}$  contain the number of occurrences of a word  $w_i$  in the context of word  $w_j$ . The matrix  $X$ , so-called the word-word co-occurrence, can be used to estimate  $P(w_j|w_i)$ , which is the probability of a word  $w_j$  to be in the context of  $w_i$ , as

$$P(w_j|w_i) = \frac{X_{i,j}}{\sum_{k=1}^{|W|} X_{i,k}}. \quad (5.34)$$

The ratios of these probabilities are relevant in estimating the relations between words. For example, the words "beverage" and "food" might occur with the same (but high) probability in the context of "refreshment", i.e.

$$\frac{P(\text{"food"}|\text{"refreshment"})}{P(\text{"beverage"}|\text{"refreshment"})} \approx 1, \quad (5.35)$$

but also occur with the same (but lower) probability in the context of "car", i.e.

$$\frac{P(\text{"food"}|\text{"car"})}{P(\text{"beverage"}|\text{"car"})} \approx 1. \quad (5.36)$$

Using such observation, Pennington et al. (2014) concluded that the context words, which give significantly different conditional probabilities, are important, e.g. for context words "eat" and "drink", i.e.

$$\frac{P(\text{"food"}|\text{"eat"})}{P(\text{"beverage"}|\text{"eat"})} > 1, \quad (5.37)$$

$$\frac{P(\text{"food"}|\text{"drink"})}{P(\text{"beverage"}|\text{"drink"})} < 1. \quad (5.38)$$

Pennington et al. (2014) formulates assumptions of the vector representations of individual words, e.g., partial interchangeability of a context word and a query word or a symmetry of the  $X$  matrix. Using these assumptions, they would like the word representations of words pairs  $f(w_i)$  and  $f(w_k)$  to satisfy

$$f(w_i)^T f(w_k) + b_i + b_k = \log(X_{i,k}), \quad (5.39)$$

where  $b_i$  and  $b_k$  are bias terms corresponding to words  $w_i$  and  $w_k$ . The conditions defined by (5.39) are not strict, thus a weighted least squares method is used, which produces the criterion function, i.e.

$$\sum_{w_i, w_k \in W} \tau(X_{i,k}) (f(w_i)^T f(w_k) + b_i + b_k - \log(X_{i,k}))^2 \quad (5.40)$$

that should be minimized. The weight corresponding to a word pair is defined by a non-decreasing function  $\tau$  which filters the frequently non-co-occurring word pairs, i.e.  $\tau(0) = 0$ , which also avoids the issue of evaluating  $\log(0)$ .

After optimizing the criterion function (5.40), the result word representations  $f(w)$  show to be more accurate in the relations between the vectors<sup>7</sup> than the results of CBOW model or skip-gram model described in Section 5.3.1 .

### 5.3.3 FastText

The methods described above, Word2vec and GloVe, assumed that words are stand-alone elements of sentences and are not structured themselves. This assumption is an issue in the case of inflected languages, where the same word can take multiple forms based on its context. To capture these properties without the need to increase training data, it is required to include the structure of words.

Bojanowski et al. (2016) proposed a different method based on mapping the character  $n$ -grams to vectors. The overall vector for a given the word is calculated as a sum of its character  $n$ -grams. More precisely, the  $n$ -grams are not chosen for a single fixed  $n$ , but for all  $3 \leq n \leq 6$ . Additionally, the method also distinguishes whether a particular character  $n$ -gram is located at the beginning of a word, in the middle, or at the end, which allows mapping prefixes or suffixes differently.

This novel approach is also capable of creating vector representations for words that were not present in the training set because the already trained character  $n$ -gram vectors can be used even for previously unobserved words to determine their representation. The benefits of this method can be seen in languages that use a multitude of compound nouns, such as German.

## 5.4 Multi-armed Bandit problem

The multi-armed bandit problem is an exemplification of the exploration-exploitation trade-off dilemma from reinforcement learning. We could introduce the definition of the problem as there are  $k$  slot machines and each has a different unknown probability distribution of rewards<sup>8</sup>, i.e., each slot machine has a different expected value of rewards. The task is to choose a slot machine to play to maximize the overall cumulative reward.

There are a few approaches on how to play on slots machines. The first straightforward approach is to play in the first round on each slot machine once, in next

---

<sup>7</sup>Find  $y'$  such that  $y$  is in the same relation to  $y'$  as  $w$  is to  $w'$  for given words  $y, w, w' \in W$ ." Then, the accuracy is measured by the amount of correctly guessed  $y'$ . Reference datasets are available and listed in Pennington et al. (2014).

<sup>8</sup>Positive reward means to win money, negative reward means to lose money.

rounds always play on the slot machine which returned the highest reward in the first round. However, the strategy used in this approach can fail, because if in the first round the slot machine with the indeed highest average reward unluckily returns a low reward, and oppositely, the slot machine with the truly lowest reward luckily returns a high reward. Thus, using this strategy, the player would always choose a slot machine with a low average reward.

The next possible approach can be more explorative. The idea is that the player would perform a lot of trials on each slot machine. And after that will choose the slot machine with the highest average reward. The disadvantage of this strategy is wasting trials on slot machines with an evidently low average reward.

This example illustrates the exploration-exploitation dilemma, in which we should balance between a greedy strategy that would exploit the options with highest estimated reward and exploring strategy that will try even less-promising alternatives that nevertheless may show to be of higher quality.

To be able to describe more advanced algorithms, we should introduce formal notation. Recall that there are  $k$  possible options. Then, for each option  $i \in \{1, \dots, k\}$  we should have two scalar values,  $n_i \in \mathbb{N}_0$  and  $\bar{r}_i \in \mathbb{R}$ , where  $n_i$  stores the number identifying how many times option  $i$  was chosen and  $\bar{r}_i$  stores the average reward among those trials, i.e.

$$\bar{r}_i = \frac{\sum_{j=1}^{n_i} r_j^i}{n_i}, \quad (5.41)$$

where  $r_1^i, \dots, r_{n_i}^i$  are the actual rewards from the individual trials of option  $i$ . The average reward might serve as an estimate of the expected value of the underlying unknown distribution of rewards. If  $n_i = 0$ ,  $\bar{r}_i$  is customarily defined as  $\infty$  for implementation purposes, so that each option is initially tried at least once.

## 5.4.1 Strategies

### Epsilon-greedy Strategy

Vermorel and Mohri (2005) evaluates that a simple  $\epsilon$ -greedy strategy works reasonably well. This strategy has a parameter  $\epsilon \in (0, 1)$  which balances between exploration and exploitation. With probability  $\epsilon$ , we choose randomly among all the  $k$  options with the uniform probability distribution. With probability  $1 - \epsilon$ , we should choose an option from the options with the highest average reward, again with the uniform probability distribution. More formally, an option  $i$  with non-maximal average reward  $\bar{r}_i < \max_{j=1}^k \bar{r}_j$  is chosen with probability

$$\frac{\epsilon}{k}, \quad (5.42)$$



whereas an option  $i$  that has maximal average reward  $\bar{r}_i = \max_{j=1}^k \bar{r}_j$  is chosen with probability

$$\frac{\epsilon}{k} + \frac{1 - \epsilon}{|A|}, \quad (5.43)$$

where  $|A|$  is the number of options with the maximal average reward, i.e.

$$A = \operatorname{argmax}_{j=1}^k \bar{r}_j. \quad (5.44)$$

In this  $\epsilon$ -greedy strategy, the value of  $\epsilon$  is fixed during the run of the whole algorithm, but it is natural to decrease its value as the sampling progresses, i.e. introduce a non-increasing sequence  $\{\epsilon_i\}_{i=1}^{\infty}$  defined, for example, as  $\epsilon_t = \min\{1, \frac{\epsilon_0}{t}\}$ , where  $\epsilon_0$  is some positive constant. Such strategies are called  $\epsilon$ -decreasing.

# Chapter 6

## Implementation

In this chapter, we would like to present our implementation of a chatbot whose implementation can be re-used to any domain (or even for multiple domains at the same time), but we have implemented it for the domain of movies. First of all, we will describe the high-level ideas and the overall architecture that is decomposed into individual modules. Second, we will talk about each module in detail. After that, we will mention some specific implementation details and explain how it was integrated into both Amazon Alexa and Slack. Finally, we will mention the used datasets.

### 6.1 Architecture Overview

On the highest level of abstraction, the chatbot can be viewed as a black-box that accepts written user text on input and produces some text output as a reply to the user query. Additionally, this black-box has internal memory so that it may produce different replies to the same query in the same conversation. To be used in an application for replying to speech, it is necessary to use some already existing techniques to convert the user speech to text and also create speech from the chatbot's reply.

In more detail, there are 6 main modules that together form the chatbot. The overall architecture in which they are arranged is shown in Figure 6.1.

First of all, when a user text is obtained, it is necessary to preprocess it before further evaluation. There are different ways of preprocessing based on into which module will the preprocessed text be sent next. One of the modules that accepts the user input is the intent classification module whose task is to determine what is the content of the user query on high level – for instance, what kind of information does the user want, whether he/she agreed to some offered information from the chatbot etc.

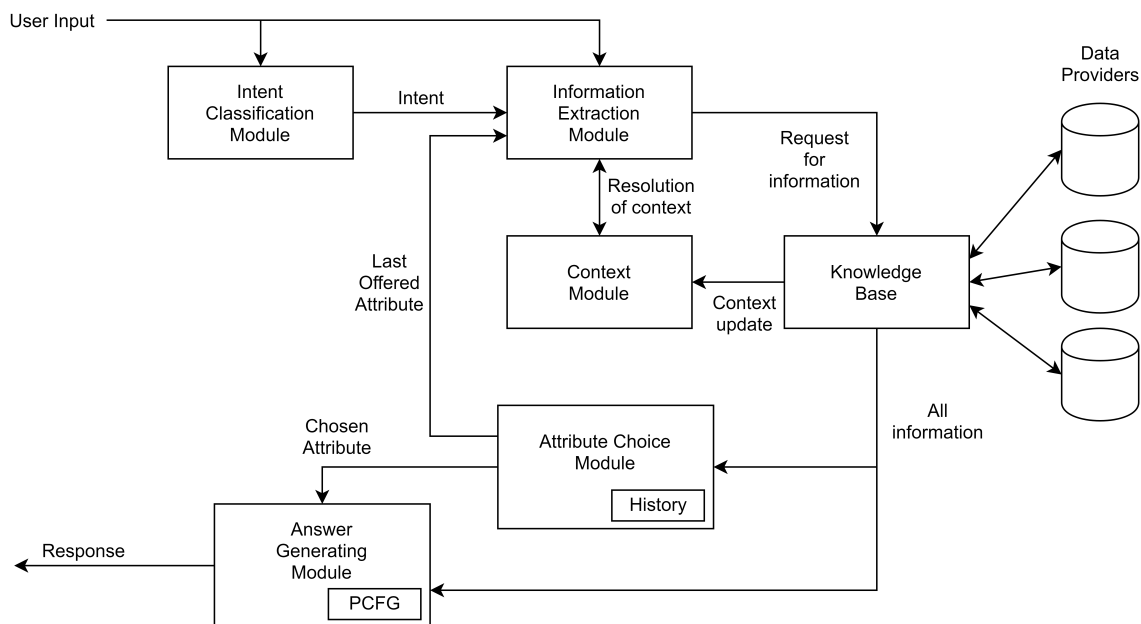


Figure 6.1: Architecture Overview.

When the intent is already known and we need to extract some information from the user query (e.g. if the user asks for the date of birth of a person, then we need to extract the person’s name), the user query is sent into the information extraction module along with a list of entities that need to be extracted. The entities throughout the whole implementation are unified to use the Schema.org<sup>1</sup> nomenclature.

As mentioned previously in the chapter describing NLP, it is necessary to also remember the context of the conversation. Context has its own module that remembers what entities are currently being discussed in order for the chatbot to be able to respond to questions from the user that use e.g. coreferences.

Now, when the intent of the user query is known and all necessary entities were recognized, we can search for the data that the user wanted in a knowledge base. Knowledge base is a part of the architecture that provides access to data sources, e.g. databases. The individual sources of data are called data providers and are unified under a single interface.

At this point, we do not have only the information from the user, but also the information that the user asked for. Based on these data, we can guess what other information (or attributes) would the user like to know and offer it to him/her. These decisions are also driven by the history of user queries and are managed by

<sup>1</sup><https://schema.org/>

the attribute choice module.

Finally, we already have all the data and also some additional reply for the user to keep the conversation fluent. These data are given to the answer generating module that forms them into a natural language response which is sent back to the user.

## 6.2 Single Building Blocks Description

We now continue with a detailed description of each module – for each of them, we will be more specific about the form of the values on the input and the output. Of course, we will also be more precise about the inner functionality of each building block.

### 6.2.1 Intent Classification Module

As the name suggests, this module is given the user-query, and its task is to determine what is the user’s intent based on the query. To do this, a recurrent neural network is utilized.

The user-written text is first converted into an appropriate form – the text is tokenized into a sequence of separated words, which are then normalized to be lower case. After that, the individual words are mapped to their vector embedding representations. For this, we have used pre-trained GloVe vectors created by the procedure outlined in Section 5.3.2. These vectors are then put into our RNN, whose last layer is softmax, so it is able to predict one of the pre-set intents. Using already trained vector embeddings is equivalent to transfer learning, in which learned parameters are taken from another already trained network and then set as fixed parameters of another one. The architecture that was chosen is described later, in Section 7.2 and is depicted in Figure 7.4 with the choice of parameters DDL3.

For our purposes, it is enough to consider only intents in which the user wants to obtain some information which is interconnected with other information in the input sentence.

Each intent is defined by two lists of entities – the first list contains what information the user gave or what is known and the second list contains what information is required by the user, in other words, what the user wants to know. For example, if the user poses the question

*”Who acted in the movie Alien?”*

then, the intent should be classified as that the user wants to know the entity

`Schema.org/actor`<sup>2</sup> for a given `Schema.org/Movie`<sup>3</sup>. Of course, the movie may not be given explicitly, because the user can ask the question

*”Who acted in it?”*

if he/she discussed the movie with the chatbot earlier. In such case, the intent is the same and does not depend on whether the information was given explicitly.

As mentioned above, there may not be only one given entity or one required entity, there can be more of both. For example, if the user asks a more general query, for example

*”Tell me something about the movie Avatar.”*

then, the required data should be a list of multiple basic attributes of a movie.

There is also an additional intent, called confirmation intent, that indicates the fact that the user is interested in the information that the chatbot offers, e.g., if the chatbot’s reply was

*”Would you like to know how old is this actor?”*

then, we should be able to detect a confirmation answer from the user. This intent is a special one because it is further processed in a slightly different way than the intents that correspond to the questions posed directly by the user. If it is detected, we ask the attribute choice module, about which we will talk in detail later, but at this point, it is important that it stores the intent of the question previously offered by the chatbot.

## 6.2.2 Information Extraction Module

Information extraction module is responsible for retrieving specified information from the user input. Usually, the task is to find the given entities in the input. The list of the given entities is already known – it was determined by the intent classification module. The task of this module is to extract each user-given entity from the input user sentence if possible.

For instance, recall the example where the user asked for an actor that appeared in a specified movie. Then, the information extraction module is given the question and the information that it should search for the entity `Schema.org/Movie`.

As shown before, it is possible that not all the entities are given in the text – for example if the user used a coreference or just agreed to obtain the information that was offered by the chatbot. In such cases, we ask the module that is responsible for context for these things. This module will be described in the next section.

---

<sup>2</sup>Shortened canonical URL for the entity actor from Schema.org ontology

<sup>3</sup>Shortened canonical URL for the entity movie from Schema.org ontology

The implementation of an information extraction module is flexible in the sense that one can easily exchange the method of extraction, which should implement a pre-defined interface. We have implemented two choices for possible use in our chatbot – the first one is the Stanford NER tagger that was trained for detecting the individual entities that are required and the second one is our approach based on a list of named entities.

The introduction of our approach was motivated by the following facts:

- Running the Stanford NER tagger on a personal computer on a given sentence with at most ten words can take up to approximately 3 or 4 seconds, which we viewed as slow because the user of the chatbot would need to wait for the reply at least for this time. We are therefore aiming for faster calculation.
- There is a lack of training data with possible user queries that would also have entities tagged by labels. For instance, if someone wanted to use our architecture for books, it is enough to provide a list of book titles that could be downloaded from already existing databases. On the other hand, to train Stanford NER, it would be necessary to find or create a list of possible user queries considering books that would also have tagged entities.
- Stanford NER may also tag elements that do not correspond to real entities. For example, if the user makes a typo and asks

*”Who directed Shawshank Redempion?”*

Stanford NER may tag ”Shawshank Redempion” as the movie in the sentence (if it is trained on a large-enough dataset previously). However, some databases may have trouble with wrongly written names of entities, that is why we would like our approach to be robust against such situations and always return an existing entity name without typos.

- Proper capitalization of named entities is also important for the Stanford NER CRF classifier, so it may lead to erroneous behavior in the case that the user would not, e.g., capitalize the name of an actor or the title of a movie. To name another example, if the chatbot would be used in Amazon Alexa, the recognized text that is formed from user’s speech may not be capitalized at all.

There are multiple ways of matching exact keywords in a sentence – for instance DFA whose graph corresponds to trie containing the words with additional transitions that are used in case that a character cannot be matched (in such case, we go to the state that corresponds to the longest suffix of the word represented by the current state). On the other hand, there are also fast algorithms for approximate comparison of a single string to another string – so-called Levenshtein automaton, which is an NFA. These procedures are described for example in Melichar et al. (2005). Running

multiple automata (that correspond to matching individual names) might be too time-consuming. That is also why we have designed a custom approach, in which we search for a substring of the user-given string and a name such that it has the minimum Levenshtein distance concerning all substrings and all names.

## User Utterance Preprocessing

When extracting information from user utterance, it is beneficial to remove the so-called **stop words**, which are words that frequently occur in a language and are therefore of low importance for categorization. E.g., in English, it might be articles or prepositions. We use a list of stop words<sup>4</sup> before performing the actual extraction. The stop words should also be removed from the different named entities in the corresponding entity lists.

To further improve the entity recognition, it is possible to extend the list of stop words by "domain-specific" stop words – in our case, we have extended the list of stop words by frequently used terms in the movie domain. For example, it is not likely that the name of an actor would contain the words "act" or "played" or that a movie title would contain the word "directed"<sup>5</sup>.

After filtering stop words, the text is then normalized to contain only lower-case letters, which is also performed on the list of given named entities. Of course, we keep copies of the original unchanged words next to the altered ones so that if we find a match among the changed words (i.e., with the removed stop words and in lower case), we return the original unchanged one.

## Proposed Method

First of all we should more formally and generally state the problem. We are given a dictionary of keywords<sup>6</sup>  $D$  that are already preprocessed. Also we have an original user utterance  $t'$  which is preprocessed to text  $t$  by the previously described procedure. Then, we would like to find a keyword  $k^*$  from  $D$  such that

$$k^* \in \operatorname{argmin}_{k \in D} \min_{s \in \delta(t)} L(s, k), \quad (6.1)$$

---

<sup>4</sup>The actual list is taken from <https://kb.yoast.com/kb/list-stop-words/>

<sup>5</sup>In fact, there exists a movie with title "Directed By" and was made in 2012 (IMDb webpage: <https://www.imdb.com/title/tt2499206/>), but as we have tested, even commercial chatbots such as Amazon Alexa are not capable of dealing correctly with such movie titles.

<sup>6</sup>For the purpose of description of our algorithm, we will use the common term "keyword" to denote the words we are interested in instead of the NLP term "named entity".

where  $\delta(t)$  are all possible  $n$ -grams<sup>7</sup> of  $t$ , i.e.

$$\delta(t) = \{t[i : j] | 1 \leq i \leq j \leq |t| \wedge (i = 1 \vee P(t[i - 1])) \wedge (j = |t| \vee P(t[j + 1]))\}, \quad (6.2)$$

where  $P$  is a predicate on characters. Formally,  $P : \Sigma \rightarrow \{true, false\}$  satisfies

$$P(c) = \begin{cases} true & \text{if the character } c \text{ is a word delimiter} \\ false & \text{otherwise} \end{cases}. \quad (6.3)$$

The following example denotes this concept

$t' =$  "Do you think you can say what movies does Naomie Harris appear in, please?"

$t =$  "think can say naomie harris appear please"

$\delta(t) = \{$  "think",  
"think can",  
"think can say",  
"think can say naomie",  
"think can say naomie harris",  
"think can say naomie harris appear",  
"think can say naomie harris appear please",  
"can",  
"can say",  
"can say naomie",  
"can say naomie harris",  
"can say naomie harris appear",  
"can say naomie harris appear please",  
"say",  
"say naomie",  
"say naomie harris",  
"say naomie harris appear",  
"say naomie harris appear please",  
"naomie",  
"naomie harris",  
"naomie harris appear",  
"naomie harris appear please",  
"harris",  
"harris appear",  
"harris appear please",  
"appear",  
"appear please",  
 $\}$

---

<sup>7</sup>A contiguous sequence of  $n$  items from a given sample of text or speech. In our case, it is a sequence of  $n$  words from the text  $t$ , where we take all  $n \in \{1, \dots, |t|\}$ .



}.

”please”

Additionally, we require that the optimal value  $\min_{s \in \delta(t)} L(s, k^*)$  should not be higher than a given maximum distance  $\bar{d}$ . The reason for this is that it would not make sense to match words with too high distance, so we should rather in such case report that there is no keyword in  $D$  that could be matched in  $t$ .

To solve this in a reasonable time, we utilize the so-called trie structure that is able to store a dictionary of words efficiently. A trie is depicted in Figure 6.2. We assume that a trie  $T$  has a root node  $T.root$  and each node in the trie (including the root) has the following functionalities:

- `node.children()` – returns the possible transitions (characters) to children nodes
- `node.child(c)` – returns the child node that is reached by transition (character)  $c$  from the node
- `node.has_word()` – returns whether the node stores a keyword
- `node.word()` – returns the stored keyword<sup>8</sup> in a node (if it stores a keyword)
- `node.has_child(c)` – returns whether there is a child node of the current node that is reachable by transitioning  $c$  (corresponds to querying whether  $c \in \text{node.children}()$ )

The trie in Figure 6.2 contains seven keywords in total, namely Allied, Alien, Aliens, Titanic, Titans, Tomb Raider, Tombstone – notice that it is also possible to store keywords with spaces. Grey nodes satisfy the predicate `has_word()`, and the transitions are written on the corresponding edges.

Such trie is used by Algorithm 2, which traverses the trie. The algorithm solves a subproblem of (6.1) – for a given text  $t$ , it searches for the keyword in trie  $T$  with the lowest Levenshtein distance to  $t$  such that the distance is not higher than  $\bar{d}$ . If there are multiple words with minimal distance, then we are interested in all of them.

The algorithm uses a priority queue (min-heap)  $q$  that contains triples  $(d, s, \text{node})$ , which correspond to search from the given node while we have already made  $d$  edit operations in the string, and the remaining part of the string that should be processed from the node is  $s$ . The key by which the priority queue is ordered is the distance  $d$ . The queue is initialized to contain distance 0, the whole input string  $t$  and to start searching from the root node. Then we proceed iteratively – until the

---

<sup>8</sup>Observe that this function may, in fact, return the original unchanged keyword before preprocessing and not the actual sequence of characters that is given by the path from the root to this node.



---

**Algorithm 2:** Algorithm that finds the closest keywords in trie  $T$  to a given string  $t$  so that the Levenshtein distance is at most  $\bar{d}$ .

---

```

1 Function find_closest_keywords( $t, T, \bar{d}$ ) is
2   best_distance  $\leftarrow \bar{d} + 1$ ;
3   best_keywords  $\leftarrow \emptyset$ ;
4    $q \leftarrow$  queue_init();
5    $q.insert(0, t, T.root)$ ;
6   while not_empty( $q$ ) do
7      $(d, s, node) \leftarrow q.pop()$ ;
8     if  $d > best\_distance$  then
9        $\_ \leftarrow$  return best_distance, best_keywords;
10    if  $|s| = 0$  then
11      if  $d < best\_distance$  and node.has_word() then
12        best_distance  $\leftarrow d$ ;
13        best_keywords  $\leftarrow \{node.word()\}$ ;
14      else if  $d = best\_distance$  and node.has_word() then
15        best_keywords  $\leftarrow best\_keywords \cup \{node.word()\}$ ;
16      else if  $d + 1 \leq best\_distance$  then
17        for transition_char in node.children() do
18           $\_ \leftarrow q.insert(d + 1, s, node.child(transition\_char))$ ;
19        continue;
20    if node.has_child( $s[1]$ ) then
21       $\_ \leftarrow q.insert(d, s[2:], node.child(s[1]))$ ;
22    if  $d + 1 \leq best\_distance$  then
23      for transition_char in node.children() do
24         $q.insert(d + 1, s, node.child(transition\_char))$ ;
25        if transition_char  $\neq s[1]$  then
26           $\_ \leftarrow q.insert(d + 1, s[2:], node.child(transition\_char))$ ;
27       $\_ \leftarrow q.insert(d + 1, s[2:], node)$ ;
28  return best_distance, best_keywords;

```

---

queue is not empty or a termination condition is not met, we pick a triple from the queue, process it and continue.

The variable `best_keywords` stores the set of keywords with the currently minimal Levenshtein distance (which is stored in `best_distance`). If we pick a triple from the queue with a distance larger than the currently best, we can terminate the search with the current result (which is optimal), because the queue contains only higher values of distance and we would also put only higher values into it.

If we have processed the whole word, then we can possibly update the currently best distance or the set of best-matching words if the current node represents a word. If it does not, we put the children nodes of the current node into the queue with distance higher by one and the same empty string – this is on line 18 in Algorithm 2 and corresponds to insert operation at the end of the word.

If we have not processed the whole word yet, we can have a look at the first character of the processed string (i.e.,  $s[1]$ ) and if the current node has a transition denoted with this character, then we can match the characters and put the rest of the string without the first node on the queue with the child of the node and no increase in distance – this is on line 21 in the pseudocode and corresponds to matching characters (i.e., no edit operation required).

However, we can also try edit operations if the distance would not get too high ( $d + 1 \leq \text{best\_distance}$ ). In such case, we can also try traversing to the child node while not changing the current string  $s$  – that corresponds to insert operation to the input string and is on line 24. We can also replace the currently first character in  $s$  by the possible transition characters that differ from it, which is on line 26. The last possible edit operation is on line 27, which represents delete of  $s[1]$  – it stays in the current node, but skips the first character of  $s$ .

Eventually, when the while loop is left, we either return the optimal (minimal) distance and the corresponding keywords or return an empty set if there were no keywords with distance lower or equal to  $\bar{d}$ .

The reason for introducing the bound  $\bar{d}$  is to avoid searching a for a keyword in a place where it is not – introducing a time bound that would terminate the while loop after some available time has passed would have a similar effect in practice.

Now, to solve the original task (6.1), we apply Algorithm 2 on each substring in  $\delta(t)$  of the preprocessed text  $t$ . Also, to increase the efficiency of evaluation on very long texts, we can filter-out too long substrings – for instance, it is not necessary to go through substrings that would be longer than the longest keyword in the dictionary. This procedure is shown in Algorithm 3.

If there are multiple keywords that have a minimum distance, we resort to using a heuristic, which is in our case taking the longest matched word.

---

**Algorithm 3:** Algorithm that finds the closest keyword in trie  $T$  to a space-delimited substring of a text  $t$  (again subject to limited maximum Levenshtein distance  $\bar{d}$ ).

---

```

1 Function find_closest_keywords_in_substrings( $t, T, \bar{d}$ ) is
2   best_distance  $\leftarrow \bar{d} + 1$ ;
3   best_keywords  $\leftarrow \emptyset$ ;
4   for  $substring$  in  $\delta(t)$  do
5     if  $|substring| > threshold$  then
6        $\lfloor$  continue;
7     distance, keywords  $\leftarrow$  find_closest_keywords( $substring, T, \bar{d}$ )
8     if distance  $<$  best_distance then
9       best_distance  $\leftarrow$  distance;
10    best_keywords  $\leftarrow$  keywords;
11    else if distance = best_distance then
12       $\lfloor$  best_keywords  $\leftarrow$  best_keywords  $\cup$  keywords;
13  return best_distance, best_keywords;

```

---

## Expected Output

In the text above, we outlined the functionality of our information extraction module. Not depending on whether we use our approach or Stanford NER to detect the entity values, the information extraction module should return a mapping of the entities to their respective values.

### 6.2.3 Context Module

As already indicated at the beginning of the previous section, a given entity may not always be present in the user text. For example, if the user answers with a confirmation intent<sup>9</sup>, then the user text is not analyzed by the information extraction module at all, and we try to derive the named entities by context. Similarly, if the user used a coreference or an anaphora, e.g.,

*"Who acted in **the movie**?"*

or

*"Who acted in **it**?"*

the information is then not present; context is queried to provide it.

Context module stores the recently mentioned named entities. Context gets updated by all the information that is extracted from the user input and also by

---

<sup>9</sup>Confirmation intent corresponds to the user accepting the information offered by the chatbot, as already mentioned in Section 6.2.1.

the information that the chatbot provides to the user. To each stored entity, context remembers how many conversational turns ago it was last mentioned. This counter is reset after any mention of the given entity, which could also be for example coreference, and increased after each turn. We have also implemented forgetting submodule which is a part of context module, which means that named entities that were not used for some long time get removed from context.

Context can be therefore viewed on a high-level as a look-up table that assigns values to mentioned named entities and removes those table entries that were not queried or updated for a long time.

## 6.2.4 Knowledge Base Module

Knowledge Base in this architecture is a module that is capable of providing information that can be obtained by combining multiple data sources. First of all, it stores a list of data providers that implement a common interface. This interface requires these providers to be able to respond whether they can provide some required data for a set of given entities. And if a data provider is capable of providing such data, it can be queried to obtain them.

For example, we might want to know the movie (i.e. the required entity) for a given name of actor. These entities are represented in the Schema.org format, so the query for a particular data provider `dp` whether it can find these data might be formulated as

```
dp.can_provide(given = [Schema.org/actor]
               required = Schema.org/Movie).
```

And if the answer is true, then we can pass a dictionary that assigns each given entity its value and query the data provider for the required entity, for example

```
dp.provide_data(given = [Schema.org/actor : "BradPitt"],
               required = Schema.org/Movie).
```

Let us remark that there could be more given entities in a single query, such as asking for the director of a particular movie, for which we know its title and the year when it was released. This would correspond to the queries

```
dp.can_provide(given = [Schema.org/Movie,
                       Schema.org/dateCreated],
               required = Schema.org/director).
```

and

```
dp.provide_data(given = [Schema.org/Movie : "Titanic",  
                        Schema.org/dateCreated : "1997"],  
               required = Schema.org/director).
```

However, such queries with multiple given entities are not detected among the possible intents due to lack of possible training data. If such data were provided, our implementation would make it possible to run such queries.

To sum it up, in each query for the data provider, we might offer multiple given entities, but we will always require only a single one. This approach allows us to utilize the data providers more flexible – it is possible that one data provider will be able to produce one of the required entities and another one of the remaining entities. In this manner, we can gradually iterate over the data providers and firstly ask whether they can assign such entities and secondly query for the value and continue until the value of the required entity is not known. In this manner, we gradually obtain the values of all the required entities if they are available and return them.

Since this approach is fairly general, we are able to implement such API for any database, because the actual querying of the data can be hidden behind it. We have implemented data providers for the IMDb<sup>10</sup>, DBpedia<sup>11</sup> and Wikidata<sup>12</sup>. The last two mentioned databases are RDF databases that were queried using SPARQL queries, whereas for IMDb, we have used a Python API<sup>13</sup>. Additionally, we have also created a data provider for prices of movies that is based on reading the data from Amazon web page directly.

In the next section, we will describe how the data were extracted from Amazon.

## Semi-structured Data Extraction

A straightforward web page was used as a representant of the semi-structured data. For this purpose we used Amazon web site<sup>14</sup>. This website provides information about movies. Specifically, we were interested in the movie sale distribution on DVD and its price.

The goal was to find a particular web page on Amazon website using Google search engine and then extract specific information from a web page which contains a record of a DVD price for a given movie title. The procedure is divided into two main blocks. The first block deals with Google searching and returning a link to a particular

---

<sup>10</sup><https://www.imdb.com/>

<sup>11</sup><https://wiki.dbpedia.org/>

<sup>12</sup><https://www.wikidata.org/>

<sup>13</sup>The package and its documentation can be found on <https://imdbpy.sourceforge.io/>.

<sup>14</sup>[www.amazon.com](http://www.amazon.com)

web page. The second block focuses on data extraction from web page content. Each block also consists of following steps. As far as the first block is concerned, firstly, the Google query with an inserted movie title was constructed. Secondly, the query was executed using standard Python external module for opening URLs and reading search results page content. Thirdly, for search results page content parsing, the content is in HTML format, the Python library, called Beautiful Soup<sup>15</sup>, was used. And finally, after parsing a link to the desired Amazon web page was returned.

We allow making one thematic digression to describe an URL structure for Amazon product links. Because we would like to extract and use some particular part of such product URL, the understanding of its structure is substantial. The certain example of Amazon product link structure can be seen in Table 6.1. The link `https://www.amazon.com/dp/B00000JLWW` refers to the web page which contains basic information of the movie "Titanic" including sale price on DVD.

Part of URL	Meaning
<code>https://</code>	HTTPS protocol
<code>www.amazon.com</code>	Amazon hostname
<code>/dp</code>	"detail product" <sup>16</sup>
<code>/B00000JLWW</code>	ASIN <sup>17</sup> , a product ID

Table 6.1: Single parts of Amazon URL

The second block tries to extract and return specific information, in particular, a movie price on DVD, from Amazon product web page. For this purpose, we would need to obtain the last part of an Amazon URL which is returned by the latest step in the first block. This last part, so-called ASIN, uniquely identifies the product. The desired Amazon web page of a product can be found using this number. After obtaining the correct product web page with the required information, the page content can be extracted. Before parsing, we have examined the page structure and its HTML code. We have found the nodes we are interested in, such as the tag identifying a movie title and the tag identifying the DVD price. To scrape these values from a web page, we have utilized XPath query language. XPath considers each HTML document to be a tree and exploits node-by-node paths.

For the detailed explanation of parsing via XPath let's take a look at the particular HTML snippet for movie title in Listing 6.1. To parse the title value from the code, we should define a location path to the desired object. This searching path

<sup>15</sup><https://pypi.org/project/beautifulsoup4/>

<sup>16</sup>Amazon specific link prefix

<sup>17</sup>Every product on Amazon site has its own unique code/number to identify it.



can look like `"/h1[@id="title"]//text()` and is processed as a query. So we are searching for `<h1>` tag from the tree root node<sup>18</sup>, where the tag has the attribute `[id="title"]`. After that children nodes of `<h1>` tag are examining, and text nodes with its values are only selected, i.e., in our case it is `<span>` tag. As a result of this procedure, the movie title with value "Titanic" will be returned.

```
<div id="title_feature_div" class="feature" data-feature-name="
  title">
  <div id="titleSection" class="a-section a-spacing-none">
    <h1 id="title" class="a-size-large a-spacing-none">
      <span id="productTitle" class="a-size-large">
        Titanic
      </span>
    </h1>
    <div id="expandTitleToggle" class="a-section a-spacing-
      none expand_aok-hidden"></div>
  </div>
</div>
```

Listing 6.1: HTML snippet for the movie title

The next snippet in Listing 6.2 represents a part of HTML code, where there is information about DVD price for a movie. Scraping of the price value was performed in the same manner as the movie title value described above. Just the location path had slightly different form `"/span[@class="a-size-base a-color-price a-color-price"]//text()`.

```
<span class="a-button-inner">
  <a href="javascript:void(0)" class="a-button-text" role="
    button">
    <span>DVD</span>
    <br>
    <span class="a-color-base">
      <span class="a-size-base a-color-price a-color-price">
        >
        $11.00
      </span>
    </span>
  </a>
</span>
```

Listing 6.2: HTML snippet for the movie price on DVD

<sup>18</sup>Tree root node is not a part of HTML document itself, it is a parent of the main document element node `<html>`.

## 6.2.5 Attribute Choice Module

The input to the attribute choice module are the two lists of named entities – the entities that were given by the user and the required entities. Based on these entities and the knowledge of the relations between them and their attributes, we should choose attributes or entities that will be present in an additional question that will help to maintain the conversation fluently.

For example, assume that the user asked for the name of the director of a particular movie, then the chatbot should not only return the name but also offer some additional information, for example,

*"Titanic was directed by James Cameron. Would you like to know what the movie is about?"*

*"Titanic was directed by James Cameron. Would you like to know some other movies from this director?"*

The output of the module is a chosen entity and its particular attribute. This pair defines a question in the already shown form mentioned in Section 6.2.1 where there is one given entity and one required entity<sup>19</sup>.

The attributes of entities that will be offered are again based on Schema.org ontology. For implementation purposes, we have selected only a subset of the attributes, because for each of them, we need a data provider. For instance, for the entity<sup>20</sup> `Schema.org/Movie`, we chose e.g. the attributes<sup>21</sup>

- `Schema.org/actor`
- `Schema.org/author`
- `Schema.org/director`
- `Schema.org/about`
- `Schema.org/dateCreated`
- `Schema.org/genre`
- `Schema.org/keywords`
- `Schema.org/inLanguage`

where the attributes `actor` and `director` are properties of `Schema.org/Movie` whereas `about`, `dateCreated`, `genre`, `inLanguage`, `keywords` and `author` are inherited from its super-class `Schema.org/CreativeWork`.

---

<sup>19</sup>In fact, it is a particular attribute for the given entity. We use the terms "attribute" and "entity" interchangeably, e.g., an actor can be an attribute of the entity movie and also a movie can be an attribute of the entity actor.

<sup>20</sup>In the Schema.org ontology, such element is called a concept or class.

<sup>21</sup>Schema.org calls these attributes "properties".

## History Submodule

A straightforward approach for the choice of attributes would be to choose randomly uniformly among all of them, but there could appear unwanted situations in which the chatbot would offer an already known or discussed attribute, which would not be useful for the user. In the most extreme case, it could happen that the chatbot would form the following response:

*"Titanic was directed by James Cameron. Would you like to know the director of Titanic?"*

which is unwanted.

That is why it is necessary to remember the history of all the answered attributes for particular named entities and also all the offered attributes that the user was not interested in. To be more clear, if the user was, e.g., not interested in the director of a particular movie, the chatbot will not offer it as an additional question again, but if the conversation later leads to a different movie, the chatbot may offer its director.

We also had to deal with the fact that not all entities are in real life disjoint in the sense that a director may also be an actor and as people, they may have shared attributes. For instance, if an answer contained Tom Hanks as a named entity of director and the user was not interested in his date of birth, we should not offer his date of birth if he is mentioned again in the role of an actor. This was resolved by matching the named entities.

Let us also remark that storing a pair of the value of the entity and its attribute in history does not forbid the user to ask for it himself or herself at a later point of time even if the user was not interested in the attribute before. The chatbot will not offer it, but if the user asks such question, the bot will answer it.

## Choice among the Remaining Attributes

At this point, we already know which attributes should not be chosen because they already appeared for the given named entity before and the task is to choose among the remaining ones. We are going to model the attribute choice using the Multi-armed bandit problem that was introduced in Section 5.4.

The options are the attributes of the individual entities, and the rewards are binary – there will be a reward equal to 1 if the user confirms that he/she is interested in the attribute and reward equal to 0 if the user is not interested. It follows certainly that we would like to offer the attributes that the user is usually interested in – i.e., have a high average reward.

More formally, there is a set of named entities  $E$  and each named entity  $e \in E$  has its set of attributes, which is denoted by  $A(e)$ . From the history submodule, we

know for each named entity  $e$  its set of already mentioned attributes  $H(e)$ ,  $H(e) \subseteq A(e)$ . The task corresponds to choosing an element from the set

$$\{(f(e), a) | e \in E, a \in A(e) - H(e)\}, \quad (6.8)$$

where  $f(e)$  is the entity of the named entity  $e$ . The choice will depend on the previously observed rewards for each entity  $g$  and its attribute  $a$ , i.e.

$$r_j^{(g,a)} = \begin{cases} 1 & \text{if user was interested in knowing the value of attribute } a \\ & \text{for entity } g \text{ when it was offered for } j\text{-th time} \\ 0 & \text{otherwise} \end{cases} \quad (6.9)$$

but we only store the average reward

$$\bar{r}_{(g,a)} = \frac{\sum_{j=1}^{n_{(g,a)}} r_j^{(g,a)}}{n_{(g,a)}} \quad (6.10)$$

and  $n_{(g,a)}$  that stores how many times the attribute  $a$  was offered in combination with the entity  $g$ .

The already introduced exploration-exploitation dilemma, in this case, translates to the trade-off between trying different attributes to improve our estimate of the attributes in which the user is interested and offering the so-far most successful attributes.

We have implemented the well-performing  $\epsilon$ -greedy strategy to make the choice of the offered entity-attribute pair based on the average reward.

As a closing remark, also observe that if the user wants to know everything about a particular thing or a person, we will eventually run out of attributes to choose from. In such a case, the attribute choice module does not choose any attribute, and a general question that is not linked to any particular entity is produced to keep the conversation flowing.

## 6.2.6 Answer Generating Module

The task of this module is to produce an answer in natural language that would contain the values that were obtained from the previously described modules, i.e.

- the given named entities that were retrieved by the information extraction module or from context,
- the required named entities that were obtained from the knowledge base based on the given named entities,

- the entity-attribute pair that we are going to offer in the additional question.

All of these objects correspond to Schema.org entities and come in a structured form into the answer generating module.

To create such answer of the chatbot, we utilize the concept of Probabilistic Context-Free Grammars to produce so-called answer templates that can be filled by the actual known information.

### Probabilistic Context-Free Grammars

Recall for example from Collins (2011) that a PCFG is defined by a set  $N$  containing non-terminal symbols, set  $T$  with terminal symbols, a start non-terminal symbol  $S \in N$  and a set  $R$  of production rules with assigned probabilities.

A production rule should be in the form

$$\alpha \rightarrow \beta_1\beta_2\cdots\beta_n, \quad (6.11)$$

where  $\alpha \in N$ ,  $n \in \mathbb{N}_0$ , and  $\beta_i \in N \cup T$  for all  $i \in \{1, \dots, n\}$ . The probabilities of production rules for an arbitrary given non-terminal  $\alpha$  should form a probability distribution, which can be interpreted as a conditional probability given the non-terminal  $\alpha$ .

An example of such grammar is in Table 6.2 that defines the production rules. The non-terminal symbols start with a dollar sign \$ and are found in the left column. The set of terminal symbols can be deduced in a straightforward way. Following this example, if  $S = \$r\_actor\_Movie$ , then we could generate for example the following responses using this grammar:

*"For instance in the movie called #Movie."*

*"#actor appeared e.g. in the movie #Movie."*

*"This person appeared e.g. in #Movie."*

These strings (called template answers) are then altered by replacing the entities tagged by the # sign by their actual values that were retrieved from the knowledge base. These tags correspond to the Schema.org ontology entities.

In the grammar which we created, there are 65 non-terminals that can in total produce 2377 individual sentences. These sentences are then further combined. For example, assume that the user asked in which movie an actor starred and as an

$\alpha$	$\rightarrow$	$\beta_1 \dots \beta_n$
<code>\$r_actor_Movie</code>	$\rightarrow$	<code>\$Actor_optional_substitution \$acted_synonyms ...</code> <code>... \$for_example in \$movie_optional_prefix #Movie.</code>
<code>\$r_actor_Movie</code>	$\rightarrow$	<code>\$For_example in \$movie_optional_prefix #Movie.</code>
<code>\$Actor_optional_substitution</code>	$\rightarrow$	<code>#actor</code>
<code>\$Actor_optional_substitution</code>	$\rightarrow$	This actor
<code>\$Actor_optional_substitution</code>	$\rightarrow$	This person
<code>\$acted_synonyms</code>	$\rightarrow$	played
<code>\$acted_synonyms</code>	$\rightarrow$	appeared
<code>\$acted_synonyms</code>	$\rightarrow$	starred
<code>\$for_example</code>	$\rightarrow$	for example
<code>\$for_example</code>	$\rightarrow$	for instance
<code>\$for_example</code>	$\rightarrow$	e.g.
<code>\$For_example</code>	$\rightarrow$	For example
<code>\$For_example</code>	$\rightarrow$	For instance
<code>\$movie_optional_prefix</code>	$\rightarrow$	the <code>\$movie_synonyms</code>
<code>\$movie_optional_prefix</code>	$\rightarrow$	the <code>\$movie_synonyms</code> called
<code>\$movie_optional_prefix</code>	$\rightarrow$	$\epsilon$ (empty word)
<code>\$movie_synonyms</code>	$\rightarrow$	movie
<code>\$movie_synonyms</code>	$\rightarrow$	film
<code>\$q_Movie_aggregateRating</code>	$\rightarrow$	<code>\$Would_you_like_to_know_synonyms</code> the ... ... rating of <code>\$movie_optional_substitution</code> ?
<code>\$Would_you_like_to_know_synonyms</code>	$\rightarrow$	Would you <code>\$also_optional</code> like to know
<code>\$Would_you_like_to_know_synonyms</code>	$\rightarrow$	Are you <code>\$also_optional</code> interested in
<code>\$Would_you_like_to_know_synonyms</code>	$\rightarrow$	Would you be <code>\$also_optional</code> interested in
<code>\$also_optional</code>	$\rightarrow$	also
<code>\$also_optional</code>	$\rightarrow$	$\epsilon$ (empty word)
<code>\$movie_optional_substitution</code>	$\rightarrow$	it
<code>\$movie_optional_substitution</code>	$\rightarrow$	the <code>\$movie_synonyms</code>
<code>\$movie_optional_substitution</code>	$\rightarrow$	<code>#Movie</code>
<code>\$movie_optional_substitution</code>	$\rightarrow$	<code>\$movie_optional_prefix #Movie</code>
<code>\$movie_optional_substitution</code>	$\rightarrow$	this <code>\$movie_synonyms</code>
$\vdots$	$\vdots$	$\vdots$

Table 6.2: Table with examples of production rules  $\alpha \rightarrow \beta_1 \dots \beta_n$ .

additional question, we will offer the rating of the found movie. This corresponds to combining the strings produced by `$r_actor_Movie` combined with the strings produced by `$r_Movie_aggregateRating`, which gives, for example, these responses:

*"This person appeared for instance in the movie #Movie. Would you be interested in the rating of #Movie?"*

*"#actor appeared, e.g., in the film #Movie. Would you be interested in the rating of #Movie?"*

*"This person appeared for example in #Movie. Are you also interested in the rating of this film?"*

Of course, such grammar does not need to be defined for each response completely independently, the used non-terminals and their production rules can be re-used for different responses.

## Used Text Templates

As indicated in Table 6.2, there are two important subsets of the non-terminals. First, those that generate an answer to a posed question by the user<sup>22</sup>, these start with `$r_` followed by identifiers of entities. Second, those that generate additional questions – these start with `$q_` again followed by the identifiers of entities that define the question. Except for these non-terminals, there are also helper non-terminals that improve the flexibility of the responses (e.g., `$acted_synonyms`) and non-terminals that are used in special cases – for example if there are no attributes to choose from, or when the knowledge base did not provide the data required by the user.

This approach of text generation belongs to the group of semi-automatic methods, because the rules for the grammar, respectively the templates require human labor to be created, but their usage and filling with data is fully automated.

## 6.3 Implementation Details

The chatbot is implemented in Python 3.6. The implementation is modular in the sense that the required functionalities are divided into classes so that each class is responsible for a single individual purpose or for decomposing a functionality and delegating its parts to multiple classes.

---

<sup>22</sup>Following on the previously shown example, `$r_actor_Movie` generates answers to the question "In which movie did a specific actor star?".

The modularity of the code allows any other developers that would like to extend or alter a single functionality to simply replace or re-implement an individual class while maintaining its interface and thus customize the bot. The code follows PEP 8 style guide for Python code and is adequately commented in all sections.

### 6.3.1 Integration to Other Systems

Our chatbot has a simple API that sends responses to the user messages. The design of its interface makes it easy to be integrated into various systems such as Amazon Alexa virtual assistant or IM platform Slack. The only requirement is to be able to redirect the messages from the users to our chatbot and then reproduce the chatbot's answer back to the users.

#### Amazon Alexa Integration

As a proof-of-concept, we have integrated our chatbot into Amazon Alexa. To do this, we created an Alexa Skill<sup>23</sup> in the Alexa Skill Kit<sup>24</sup> called MovieChatBot that we have connected to an AWS Lambda Function<sup>25</sup> created by us.

Amazon Alexa accepts written or spoken input – in case of spoken input, it is transformed into its written form and further processed in the same manner. After our skill is invoked (for example, by the user saying "run MovieChatBot"), the user inputs are processed by intents<sup>26</sup>. The intents are usually identified by hand-written sample queries that serve as a template for what the user might say to invoke a particular intent. This usually requires a comprehensive list of possible requests. If we wanted to implement our chatbot in Alexa, we would, for example, need to define a AskForDirectorOfMovieIntent with the following user queries defined:

*Who directed {Movie}?*  
*Who is the director of {Movie}?*  
*Could you tell me who directed {Movie}?*  
*Would you know the director of {Movie}?*

where {Movie} is called a slot that can take various pre-defined values and is returned to be processed further along with the categorized intent.

This is however not necessary in our case, because we have already implemented such categorization. For us, it is only necessary to define a single slot, which we called

---

<sup>23</sup>An Alexa Skill can be roughly viewed as a conversational programme for Alexa.

<sup>24</sup>The developer webpage for Alexa Skill Kit is [developer.amazon.com](https://developer.amazon.com), where one can create the, for example, the intents and custom slots or additionally build and test a created skill.

<sup>25</sup>The webpage where lambda functions can be set-up is [console.aws.amazon.com](https://console.aws.amazon.com)

<sup>26</sup>These intents are intents in the Amazon Alexa nomenclature and our implementation do not correspond to our previously mentioned classes of intents.



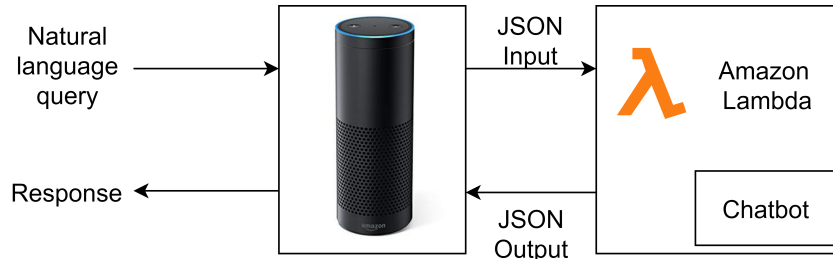


Figure 6.3: Amazon Alexa Integration Schema.

{Query} and is of type `AMAZON.SearchQuery`. This type of slot can match an arbitrary phrase, which in our case will be the user query<sup>27</sup>.

The content of the user query is then forwarded to our Lambda Function, which just extracts it from the obtained JSON, calls the API of our chatbot to produce a response and send an answer that contains the response again in the JSON form. Alexa then processes this response and shows the answer to the user either in text mode or as a speech, depending on the mode of conversation.

## Slack Integration

We have also integrated our chatbot into Slack by creating a Slackbot in our Slack App. We have then created functionality that serves as a client that responds to user queries if the chatbot is mentioned at the beginning of the query as usual in Slack. For readers that do not know Slack, this means sending, for example, the message

*"@MovieChatBot who directed Titanic?"*

Our chatbot can be added into multiple channels at once and will keep the contexts separated for each channel. Each time it is added to a new channel, it will greet its users.

<sup>27</sup>Let us remark a little implementation detail – Alexa Skill Kit does not allow only a slot to be present as a template for some intent, at least one fixed word needs to be present too. This can be overcome by, e.g., calling the chatbot by its name in each query. The retrieval of the whole actual user query given to Alexa is more complicated.

### 6.3.2 Keras

To implement the neural network, we have used the Python implementation of Keras package that offers a wide variety of options for modeling a neural network and its learning. We have modeled our RNN architectures using this package that was also used for training.

## 6.4 Datasets Description

Throughout this implementation, we make use of multiple datasets that allowed us to implement different parts of the chatbot. On top of the datasets from other parties, we have also created our data as an extension of the currently existing datasets.

### Movie Dataset

This dataset is from HPI (2017), which is a dataset extracted from IMDb that consists of lists of many structured movie information, e.g., list of movie titles, list of actor/director/author names, list of movie keywords. From this dataset, we use the lists of movie titles, actors, actresses, directors, writers and keywords. We extract the data and preprocess so that they can be used for named entity recognition by the previously presented procedure in user-written queries.

### WikiMovies Dataset

To train our intent classifier to predict the category of user's question and also for Stanford NER training, we used the WikiMovies Dataset, which is a part of the bAbI project<sup>28</sup>. This dataset contains a list of questions and their corresponding answers that can be divided into 13 categories. From these datasets, we use only the questions.

The categories are the following:

- to an actor/actress assign a movie in which he/she starred
- to a director assign a movie that he/she directed
- to a movie assign an actor that acted in it
- to a movie assign its director
- to a movie assign its genre

---

<sup>28</sup>The dataset can be downloaded here: <https://research.fb.com/downloads/babi/>.

- to a movie assign its IMDb rating
- to a movie assign how many people have rated it
- to a movie assign its language
- to a movie assign its screenwriter
- to a movie assign its release date
- to a keyword assign a corresponding movie
- to a screenwriter assign a movie written by him
- to a movie assign its tags

## Testing Datasets

To be able to test the approaches on previously unseen data, we have also extracted the same lists of named entities from Wikidata by SPARQL queries, except for a list of keywords that are not available on Wikidata, so they were taken from MovieKeywords (2018).

## Our Generated Dataset

After using the WikiMovies dataset, we have concluded that it may not be sufficient enough and that the listed questions do not generalize too much in practice. That is why we decided to extend this dataset initially containing 149 question templates (which we had to extract manually) by additional 61 templates created by us to improve the performance of not only the intent classifier but also the trained Stanford NER implementation. We have also introduced a new text classification category for the confirmation of the questions offered by the chatbot and exchanged the last category for a general question on the information about a movie because they had significant semantic overlap.

Moreover, we have also created several possible introductions texts that can be combined with almost any question, such as "Could you tell me" or "Tell me." This has also added more variance to the length of the possible inputs. This created in total 2400 unique templates from which we created the actual training dataset for our RNN by replacing the template words by randomly generated entities from the Movie Dataset. For training Stanford NER, only 2388 unique templates were used as we omit the question confirmation category, where no entity extraction is performed.

# Chapter 7

## Experiments

In this chapter, we first evaluate the proposed methods – i.e., compare two mentioned methods of NER and also present our experimental results of intent classification based on LSTM. Second, we show examples of the dialogs of the chatbot with users.

### 7.1 NER comparison

To compare the two implemented named-entity recognition systems, we needed to create two separate datasets – one with training data and one with testing data. These two datasets must be created carefully to avoid disadvantaging one system as compared to the other – especially in this case because the output of the method based on the trie is mainly determined by the list of named entities given to it, whereas the Stanford NER trains from the structure of whole tagged sentences.

#### 7.1.1 Train and Test Setting

That is why we utilized multiple different datasets during training and testing. First of all, the user question templates were divided into four folds which will be used for cross-validation. Then, we filled the missing slots in the sentences in the folds by two different procedures:

- if the fold is to be used in training, the source of all named entities is IMDb
- if the fold is to be used in testing, the source of the named entities is Wikidata, except for the set of keywords that was taken from a different source, as discussed in Section 6.4.

In both cases (i.e., training and testing folds), the generated sentences are not always properly capitalized, because as said before, we can not rely on capitalized input. Therefore, in 50 % of the sentences, the named entities are properly capitalized, whereas, in others, they are not. The same holds for the capitalization of the first letter of the sentence.

So, unlike classic  $k$ -fold cross-validation that uses only  $k$  sets of data, we need to use  $2k$  sets of data. The reason for this is that the trie method will be trained on whole lists of named entities from IMDb, whereas Stanford NER classifier will always be learned from some three folds of annotated training data which contain user questions with named entities from IMDb. Then, to test both approaches, they are both given the same sentences from the fourth fold of testing data. For Stanford NER, it is important that the templates on which it is tested are different, because it learns the structures of sentences, therefore they can be viewed as unobserved. For trie method, it is important that the named entities contained in the testing fold are taken from different dataset than from which it was trained.

The sizes of the individual training sets in cross-validation for Stanford NER are in Table 7.1, where we by "Original templates" mean the number of templates before various introductions or endings are applied to them. Templates with introductions and endings are called "Extended templates" in which the slots were filled by different entities to obtain the overall amount of training sentences. The sizes of the datasets used to create the tries for individual entities are in Table 7.2.

The sizes of the individual testing folds are in Table 7.3 which follows the same notation as Table 7.1.

<b>Entity</b>	<b>Original templates</b>	<b>Extended templates</b>	<b>Sentences</b>
actor	17–18	204–216	40800–43200
author	9–10	108–120	21600–24000
movie	103–104	1236–1248	61800–62400
director	12–13	144–156	28800–31200
keywords	6	72	14400

Table 7.1: Overview of characteristics of the training data for Stanford NER (for each fold).

<b>Entity</b>	<b>Number of named entities</b>
actor	1018040
author	194996
movie	1361188
director	151635
keywords	236625

Table 7.2: Overview of characteristics of the given lists of data to the trie.

Entity	Original templates	Extended templates	Sentences
actor	5–6	60–72	1200–1440
author	3–4	36–48	720–960
movie	34–35	408–420	1224–1260
director	4–5	48–60	960–1200
keywords	2	24	480

Table 7.3: Overview of characteristics of the testing data (for each fold).

## 7.1.2 Results

The named entity recognition systems were trained and tested on each required entity separately because the information extraction module is queried already with the knowledge of the entity that should be searched. Training Stanford NER for all entities at once may disadvantage it as it would also try to tag different entities.

The results are shown in Table 7.4. The table also contains the overall amount of sentences on which it was tested (when all testing samples overall folds are aggregated).

Entity	Cumulated test set size	Trie method	Stanford NER
actor	5520	0.94	0.89
author	3120	0.83	0.85
movie	4968	0.69	0.79
director	4080	0.90	0.88
keywords	1920	0.69	1.00

Table 7.4: Comparison of two methods for NER by their performance.

The performance of each model was calculated as the ratio of correctly identified named entities over the size of the test set. To decide whether the found named entity corresponds to the ground truth, we did not care about

- whether the definite article "the" was used or not, e.g., if the original question was "Who directed The Avengers?", we accept "Avengers" as correct recognition.
- whether the name was correctly capitalized or not, e.g., an answer "avengers" would also be correct to the previous example.
- whether the name contained punctuation, e.g. the keyword "CIA" will also be accepted as "C.I.A."
- white space characters

Additionally, to neglecting these details, we also allow a single edit operation, so that we allow the Levenshtein distance between ground truth and the recognized entity to be at most 1. This is for the purposes of possible different spellings or almost identical keywords (e.g. "Miloš Forman" vs "Milos Forman" or "doctor" vs "doctors").

On the entities actor, author, and director, both methods achieved similar and good results whereas, for movie and keywords entities, Stanford NER achieved significantly better results.

In keywords, the poor performance of the trie method was caused by the fact that any word or a combination of words can become a keyword. That is why it was hard to match it to the already existing ones. However, mistakes were often caused by tagging only a part of the keyword, e.g., tagging "chinese" instead of "chinese new year" or "sledding" instead of "dog sledding". On the other hand, Stanford NER did not make any mistakes on this dataset – we believe that it was caused by too small amount of templates that could be resolved even by a simple rule-based algorithm.

In movies, the lowered performance was caused by the fact that words used in movie titles may commonly appear as usual words in a text. We could also explain this from the other point of view. That is, why the names of the people performed well. We think that it is because names of people are not strings of characters that would be in a text and not denote a person. That is why the error on the names of people was low.

In general, the kinds of mistakes of both classifiers were significantly different in the structure of the wrongly classified sentences. In the case of Stanford NER, it sometimes happened that it systematically tagged a wrong additional word into the named entity, e.g., in the sentences

*"can you tell me which language can I hear in Fight Club"*

*"Can you tell me which language can i hear in buffy the vampire slayer"*

*"Tell me how would people rate iron man"*

it detected the words

*"I hear in Fight Club"*

*"i hear in buffy the vampire slayer"*

*"rate iron man"*

as the named entities. Another issue of Stanford NER was that it sometimes did not match any word in the sentence as the actual named entity.

On the other hand, the trie method had a different kind of errors – it tagged a

different part of the sentence to be the actual named entity, e.g., in the sentence

*"can you tell me when was The Blob made please"*

it recognized *"please"*. Or, it tagged only a part of the whole movie name, e.g., it detected *"star wars"* in the sentence

*"Tell me when was star wars episode iii: revenge of the sith made please"*

but the tagged named entity should have been *"star wars episode iii: revenge of the sith"*.

### 7.1.3 Runtime

The number of tested sentences was limited mainly by the runtime of Stanford NER, which was not so fast in classifying them. The average runtime for each entity during the testing phase is overviewed in Table 7.5.

Entity	Trie method	Stanford NER
actor	0.084 sec.	1.5 sec.
author	0.091 sec.	1.2 sec.
movie	0.056 sec.	2.7 sec.
director	0.076 sec.	1.3 sec.
keywords	0.070 sec.	0.77 sec.

Table 7.5: Comparison of two methods for NER by their average runtime per tagged sentence.

## 7.2 Intent Classification

Recall from Section 6.2.1 that for intent classification, we use a recurrent neural network. For purposes of testing and choosing the right hyper-parameters, we have created the training, validation and testing sets in the following manner:

1. divide all the original templates into three sets that will later correspond to training, validation and testing
2. for each set separately, extend the templates by introductions and endings as in the previous section.
3. instantiate the sentences by filling slots by named entities



We approach the last step similarly as in the previous case of NER. For training and validation, the named entities are sampled from a different data source than in the testing set. Based on the complexity of the network, we designed five architectures that differ by the number of layers. In each of them, LSTM cell was present. The sizes of datasets are shown in Table 7.6.

Dataset	Original templates	Extended templates	Sentences
Training	152	1724	18240
Validation	38	446	4560
Testing	20	240	2400

Table 7.6: Sizes of datasets for NN training and evaluation.

In this case, as opposed to NER experiments, we also incorporate the confirmation templates. But, these can not be used with the general introductions, so we have only repeated them to have an approximately same amount of samples from all classes throughout the sets. Without doing this, the trained NN showed worse results for the confirmation class.

In each case, we trained the network on the generated training data and used early stopping calculated on the validation data to avoid overfitting. The accuracy was evaluated after each training epoch. With networks that contained fully connected (dense) layers, we also applied dropout.

### Simple LSTM cell

First of all, we tried the most simple architecture, which is depicted in Figure 7.1. In this architecture, each word on the input is mapped to its vector representation using pre-trained Glove vectors<sup>1</sup>, which is directly sent to the LSTM cell. Part of the output of the cell is then sent back to it with the next word etc. With the last word, we put the output of the LSTM cell into the softmax layer that classifies among the 14 possible intents.

We have evaluated five networks of this form that differed by the dimension of Glove vectors on the input and also by the used activation function in the LSTM. The parameters of the models are in Table 7.7.

---

<sup>1</sup>The vectors were downloaded from <https://nlp.stanford.edu/projects/glove>, where datasets with vector dimensions of 50, 100, 200 and 300 are available.

Model id	DE	AL	Validation accuracy
L1	200	tanh	<b>0.848</b>
L2	200	ReLU	0.796
L3	100	ReLU	0.761
L4	50	tanh	0.800
L5	300	tanh	0.792

Table 7.7: Results on simple LSTM cells depicted in Figure 7.1.

Model id	DE	AL	DL	D	VA
LD1	200	tanh	50	0.5	0.902
LD2	200	tanh	20	0	0.871
LD3	100	ReLU	40	0.5	0.871
LD4	50	tanh	100	0.25	0.814
LD5	300	tanh	150	0.75	<b>0.939</b>
LD6	300	ReLU	150	0.75	0.893

Table 7.8: Results on LSTM cells followed by dense layer depicted in Figure 7.2.

Model id	DE	A	D1	DL	D	VA
DLD1	100	tanh	100	50	0.5	0.928
DLD2	200	ReLU	150	150	0.5	0.932
DLD3	200	ReLU	250	100	0.25	0.929
DLD4	300	ReLU	200	75	0.5	0.940
DLD5	300	tanh	200	150	0.75	0.927
DLD6	300	ReLU	200	150	0.75	<b>0.944</b>
DLD7	300	tanh	200	75	0.5	0.927
DLD8	100	ReLU	100	50	0.5	0.935
DLD9	200	ReLU	100	75	0.75	0.939

Table 7.9: Results on architecture depicted in Figure 7.3.

Model id	DE	D1	D2	DL	D	VA
DDLD1	200	150	100	100	0.5	0.953
DDLD2	300	280	180	220	0.75	0.952
DDLD3	200	100	100	50	0.25	<b>0.960</b>
DDLD4	300	300	300	300	0.5	0.953
DDLD5	300	250	100	150	0.25	0.940

Table 7.10: Results on architecture depicted in Figure 7.4.

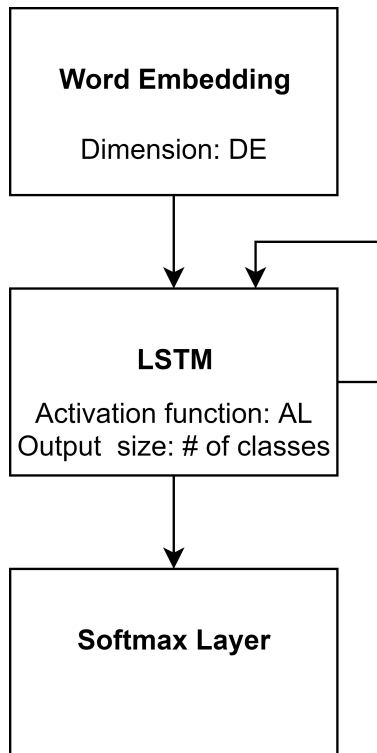


Figure 7.1: Simple LSTM cell variant.

Model id	DE	D1	D2	DL	D3	D	VA
DDLDD1	200	100	100	100	50	0.25	0.941
DDLDD2	300	200	150	100	75	0.5	0.962
DDLDD3	300	200	100	75	100	0.5	<b>0.966</b>
DDLDD4	200	250	200	150	75	0.25	0.942
DDLDD5	200	150	100	300	150	0.75	0.903

Table 7.11: Results on architecture depicted in Figure 7.5.

### Shallow RNN with Dense Layer Before Output

Because the previous results were not convincing enough, we decided to add a layer between the LSTM cell and the softmax. This allowed changing the dimension of the inner state of LSTM. We have also added dropout between the LSTM and dense layer, as depicted in Figure 7.2. We evaluated 6 models listed in Table 7.8.

From the results, we concluded that even for a small classification problem like ours, it might be beneficial to use larger models. Also, we noticed the inferior behavior of ReLU as compared to tanh when used in LSTM cell. That is why in further experiments, we will only use tanh. We have also ruled out using only 50-dimensional Glove vectors.

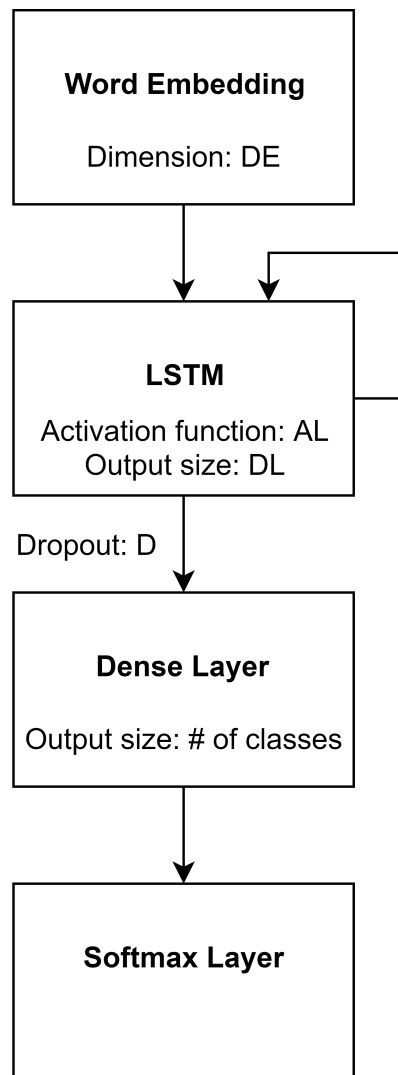


Figure 7.2: Shallow RNN with dense layer.

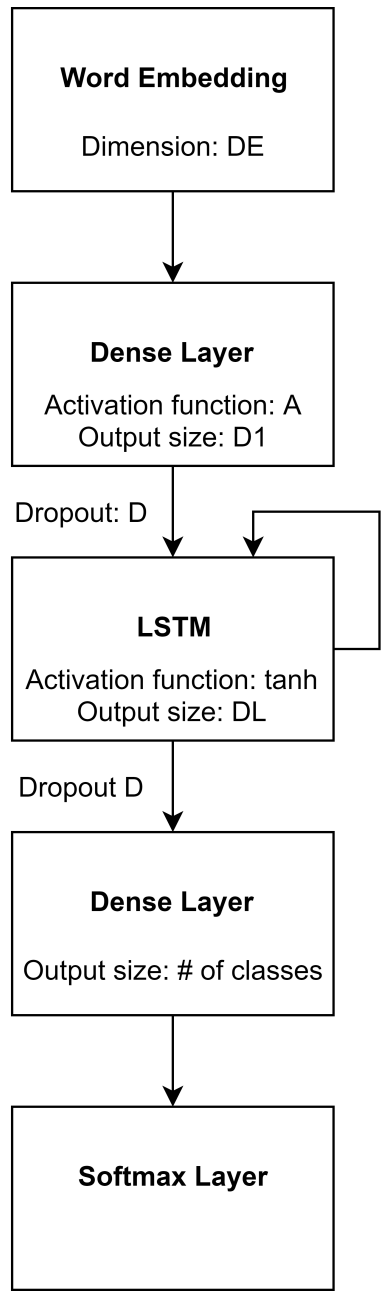


Figure 7.3: RNN with two dense layers.

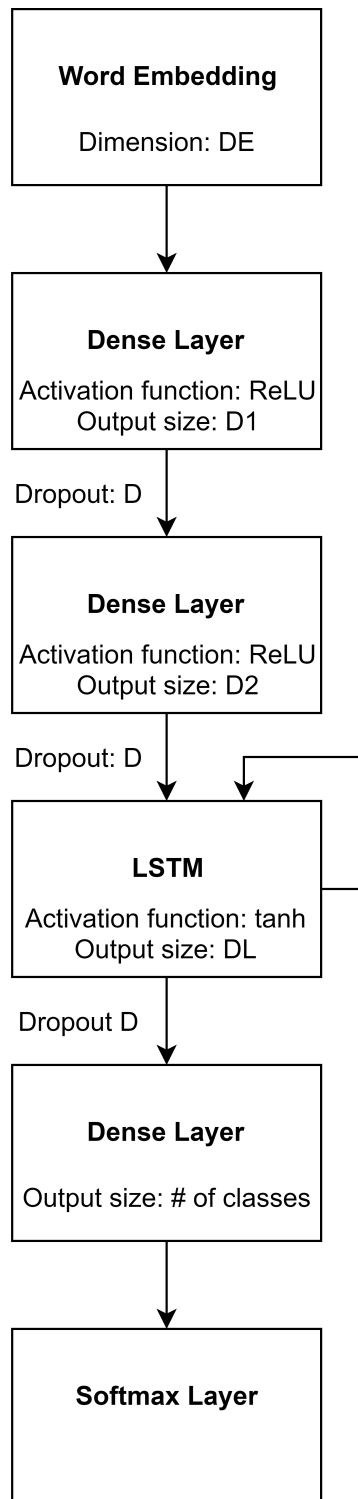


Figure 7.4: RNN with three dense layers.

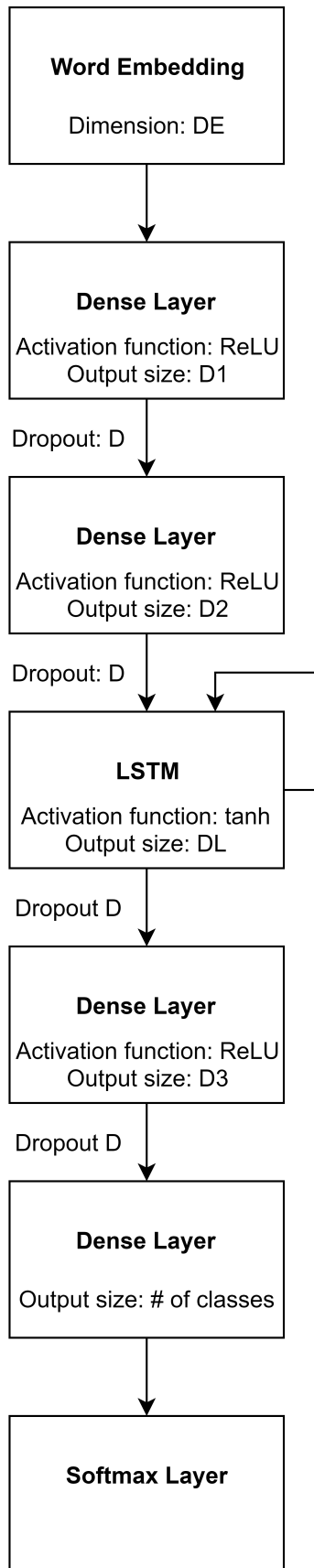


Figure 7.5: RNN with four dense layers.



Figure 7.6: Accuracy of chosen models on a validation set during training.

## RNN with Dense Layers on both Input and Output

In this case, we have also added one fully connected layer with an activation function (and also with dropout) before the LSTM cell. We considered tanh and ReLU activation functions and tested 5 different models whose parameters are listed in Table 7.9 and correspond to Figure 7.3.

We observed that tanh activation function is inferior to ReLU when used after the fully connected layer<sup>2</sup> and that the larger models still yield good results.

## RNN with Multiple Following Dense Layers

Eventually, we have also tried to have two fully connected layers with ReLU activation functions followed by an LSTM cell behind which is a third fully connected layer. This architecture is depicted in Figure 7.4 and the corresponding results are in Table 7.10.

Because this architecture again improved the validation accuracy, we decided to add one more dense layer, as depicted in Figure 7.5. The results of this architecture

<sup>2</sup>This is why tanh is not considered in the next models.



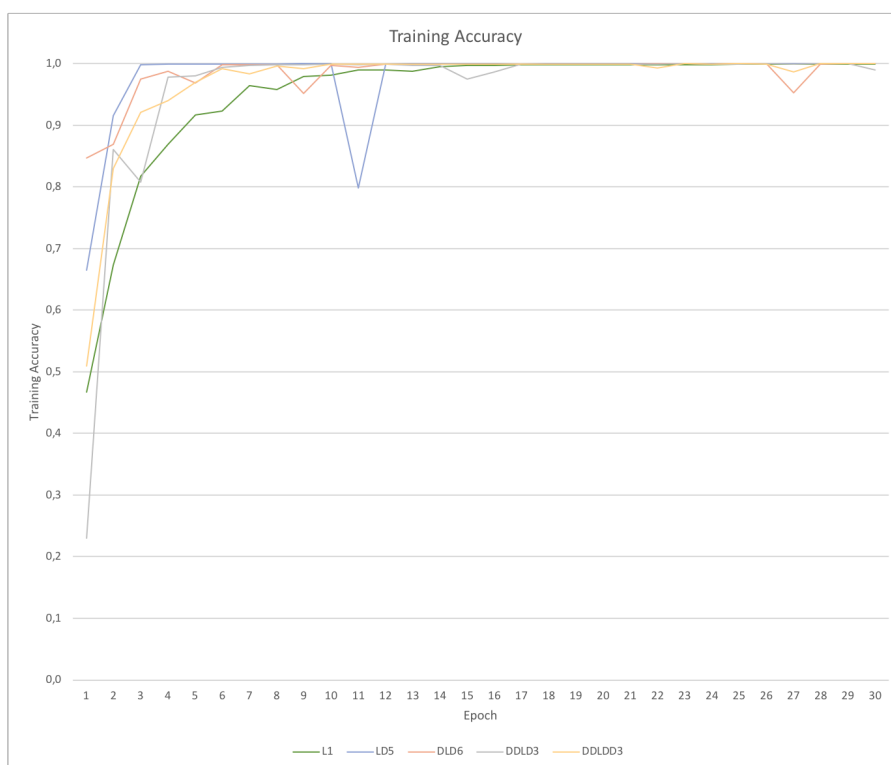


Figure 7.7: Accuracy of chosen models on a training set during training.

are in Table 7.11 and it can be seen that there has been only little improvement as compared to the previous architecture.

Thus, we conclude that it may already be too large for our classification problem and return to the previous architecture, where the best-performing model was DDLDD3. We tested this model on the previously unused testing data and reached accuracy **0.910**.

In Figure 7.6 we can see accuracy on the validation set and in Figure 7.7 – accuracy on the training set for the best model from each architecture with respect to current epoch.

### 7.3 Simple Dialog Examples

In this section, we present simple dialogs that were conducted with users. While performing the dialogs, we stored a log that captured the whole process in the background – e.g., the result of intent classification, NER, knowledge base querying, attribute choice and the process producing an NL response. Using this log, we can

evaluate the dialogs after they were already performed to detect possible causes of misunderstandings.

In all cases, the users of the chatbot were informed which information it can provide along with the assumption on its usage. This means that the user is assumed to be constructively cooperative and remain in the domain of movies. The user can:

- ask one of the possible expected questions,
- ask for general information considering a movie,
- accept the information that the bot is offering,
- decline the information that the bot is offering but ask another question instead.

On the other hand, the user can not ask questions outside the domain or be uncooperative in the sense that the user would for example decline what the bot is offering and not say what he/she is interested in.

The first four dialogs were conducted using the trie method whereas the fifth dialog uses Stanford NER method. We will now list the full dialogs and comment on their mistakes.

### 7.3.1 Dialog 1

1: Chatbot: Hello there, I am your chatbot for movies. What would you like to know?

2: User: hey tell me smth about se7en

3: C: Se7en was made by David Fincher in the year 1995 and has an exciting plot . Two detectives, a rookie and a veteran, hunt a serial killer who uses the seven deadly sins as his motives. Would you also like to know the genre of Se7en ?

4: U: yeah

5: C: That is a crime. Are you also interested in who starred in it ?

6: U: hm ok

7: C: For instance Morgan Freeman. Would you like to know the rating of this film ?

8: U: yes

9: C: Se7en has rating of 8.6. Would you like to know the price of this film ?

10: U: ok

11: C: You can purchase this movie on Amazon for \$16.77. Would you like to know the language of it ?

12: U: no tell the genre of se7en

13: C: This film is a crime. Would you also like to know the screenwriter of it ?

14: U: no tell me other movies where morgan freeman acted

15: C: This person played e.g. in the movie called 10 Items or Less. Would you like to know who made the movie ?

16: U: yeah tell me about it

17: C: The movie Sepah: The Movie was made by Rosli in the year 2012 and has an exciting storyline . Three best friends gamble with one another to take a challenge to win over a beautiful girl that works in an electronic industry. Would you be interested in some other movies made by this director ?

18: U: no tell me other movies where morgan freeman acted

19: C: For instance in Attica. Are you interested in when was Morgan Freeman born?

20: U: yea

21: C: Morgan Freeman was born on 1937-06-01. Would you also like to know something else ?

22: U: tell other movies with morgan freeman

23: C: This person starred for example in the film called Fight for Life. Would you also like to know the genre of the movie Fight for Life ?

24: U: no tell other movies with morgan freeman

25: C: This person played e.g. in Edison. Are you interested in the screenwriter of it ?

26: U: no tell other movies with morgan freeman

27: C: Morgan Freeman played e.g. in the movie Olympus Has Fallen. Would you be also interested in the rating of the movie ?

28: U: yes

29: C: Rating of the film called Olympus Has Fallen is 6.5. Would you be interested in the price of Olympus Has Fallen ?

30: U: no tell more about this movie

31: C: Olympus Has Fallen was made by Antoine Fuqua in the year 2013 and has an exciting plot . Disgraced Secret Service agent (and former presidential guard) Mike Banning finds himself trapped inside the White House in the wake of a terrorist attack; using his inside knowledge, Banning works with national security to rescue the President from his kidnappers. Would you also like to know when was this director born?

32: U: yes

33: C: Antoine Fuqua was born on 1965-05-30. Would you also like to know some other movies directed by this director ?

34: U: ok

35: C: This director made for instance the film The Replacement Killers. Would you like to know the screenwriter of it ?

36: U: yes

37: C: This film is based on a screenplay from Ken Sanzel. Are you interested in some other movies written by this screenwriter ?

38: U: ok

39: C: Ken Sanzel wrote for example the script for the film NYC 22. Would you like to know when was this screenwriter born?

We evaluate this dialog as mainly successful with the only error in understanding the user query on line 16 where the chatbot misclassified the intent of the user utterance and classified it as general information about a new movie. The movie title was extracted from the user input also by mistake.

### 7.3.2 Dialog 2

1: Chatbot: Hello there, I am your chatbot for movies. What would you like to know?

2: User: Hello, Sir, could you tell me something about the movie Last Stand?

3: C: The movie was made by Lloyd A. Simandl in the year 2000 and has an interesting plot . A small band of desperate freedom fighters must topple a vicious dictator before he destroys mankind's last hope. Would you be interested in the screenwriter of the film ?

4: U: Yes, please.

5: C: This film is based on a screenplay from Chris Hyde. Are you interested in who starred in the film ?

6: U: Why not!

7: C: Kate Rodger starred in it . Are you also interested in the rating of it ?

8: U: Sure.

9: C: Rating of the film Last Stand is 3.6. Would you like to know how many people have rated the movie ?

10: U: No, the movie is not good. Could you tell me whether Terminator is good?

11: C: The film Terminator has rating of 8.0. Would you be also interested in the price of Terminator ?

12: U: No, tell me its plot, please.

13: C: Rating of Please is 6.9. Would you be also interested in its plot ?

14: U: No. I want to know what is Terminator about.

15: C: I am sorry, we did not find the required data in our system. Are you also interested in the genre of the movie If You Want to Know Who We Are ?

16: U: No. Tell me finally something about Terminator!

17: C: Terminator was directed by James Cameron in the year 1984 and has an engaging storyline . A seemingly indestructible android is sent from 2029 to 1984 to assassinate a waitress, whose unborn son will lead humanity in a war against the machines, while a soldier from that war is sent to protect her at all costs. Would you like to know the language of Terminator ?

Even in this second dialog misclassification of the user input happened. On line

12 the chatbot detected a wrong intent again with with a wrong movie title. The user tried to correct the chatbot, but was unsuccessful until the third try.

### 7.3.3 Dialog 3

1: Chatbot: Hello there, I am your chatbot for movies. What would you like to know?

2: User: Lets talk about Gran Torino. What can you tell me about it?

3: C: Gran Torino was directed by Clint Eastwood in 2008 and has an fascinating plot. Disgruntled Korean War veteran Walt Kowalski sets out to reform his neighbor, a Hmong teenager who tried to steal Kowalski's prized possession: a 1972 Gran Torino. Are you also interested in the rating of the movie ?

4: U: yes please

5: C: Rating of Gran Torino is 8.1. Would you like to know who starred in this movie?

6: U: Sure

7: C: Christopher Carley appeared in Gran Torino. Would you like to know when this person was born?

8: U: No. Which other movies were directed by Clint Eastwood?

9: C: Clint Eastwood directed for example Breezy. Would you like to know the screenwriter of this movie?

10: U: No, but could you tell me more about this movie?

11: C: Breezy was directed by Clint Eastwood in 1973 and has an exciting plot. Breezy is a teen-aged hippy with a big heart. After taking a ride with a man who only wants her for sex, Breezy manages to escape. She runs to hide on a secluded property where stands the home of a middle-aged divorced man, Frank Harmon. Frank reluctantly takes Breezy in only to fall, unexpectedly, in love with her. Would you like to know the genre of this movie?

12: U: yes

13: C: Breezy is a drama. Would you like to know how many people have rated this movie?

14: U: no, i would rather know its rating

15: C: Rating of I Would Rather is 8.5. Would you like to know who directed this movie?

16: U: no, who directed breezy

17: C: It was Clint Eastwood's work. Would you like to know when was Clint Eastwood born?

18: U: no i want to know who acted in it

19: C: I am sorry, we did not find the required data in our system. Would you like to know its plot?

20: U: no, what is the rating of this movie?

21: C: Rating of Breezy is 7.0. Would you like to know who starred in this movie?  
 22: U: absolutely  
 23: C: For example, William Holden starred in Breezy. Would you like to know some other movies with William Holden ?  
 24: U: maybe  
 25: C: William Holden appeared for example in Texas. Would you like to know who directed Texas ?  
 26: U: yes, tell me that  
 27: C: Michael Gordon directed Texas. Would you like to know some other movies directed by this person?  
 28: U: tell me something about Mr. & Mrs. Smith  
 29: C: Mr. & Mrs. Smith was directed by Doug Liman in 2005 and has an exciting plot. A bored married couple is surprised to learn that they are both assassins hired by competing agencies to kill each other. Would you like to know who starred in this movie?  
 30: U: yes please  
 31: C: Vince Vaughn appeared in Mr. & Mrs. Smith. Would you like to know the rating of this movie?  
 32: U: Who else starred in it?  
 33: C: Jack O'Connell appeared in Starred Up. Would you like to know when this person was born?

The dialog performed well, until the line 14, where the chatbot detected a wrong movie title in the user utterance, but was able to return to the original context after a few dialog turns. A similar mistake happened on line 32, where the intent was classified correctly, but a wrong named entity was detected.

### 7.3.4 Dialog 4

1: Chatbot: Hello there, I am your chatbot for movies. What would you like to know?  
 2: User: Tell me a movie that was written by Christopher Nolan.  
 3: C: This screenwriter wrote for example the script for the film The Dark Knight. Would you like to know the rating of The Dark Knight ?  
 4: U: Yes.  
 5: C: Rating of the movie called The Dark Knight is 9.0. Would you be also interested in who made it ?  
 6: U: No. What is language of movie?  
 7: C: The language of The Dark Knight is English. Would you also like to know its storyline ?  
 8: U: Sure.

9: C: When the menace known as the Joker emerges from his mysterious past, he wreaks havoc and chaos on the people of Gotham. The Dark Knight must accept one of the greatest psychological and physical tests of his ability to fight injustice. Would you also like to know the price of it on Amazon ?

10: U: No. Who is its director?

11: C: Christopher Nolan directed The Dark Knight . Would you also like to know when the movie called The Dark Knight was filmed ?

12: U: Maybe

13: C: The Dark Knight was shot in the year 2008. Are you also interested in how many people have rated it ?

14: U: Absolutely yes!

15: C: The Dark Knight has 2004684 ratings. Would you be interested in the genre of it ?

16: U: No, what is rating of movie?

17: C: Rating of the movie The Dark Knight is 9.0. Would you like to know something else?

18: U: Who acted in it?

19: C: For instance Heath Ledger. Would you like to know some other movies with Heath Ledger ?

20: U: Yes.

21: C: This actor appeared for instance in Clowning Around. Would you like to know when was this person born?

22: U: No, could you tell me about the film Twilight?

23: C: Twilight was made by Catherine Hardwicke in 2008 and has an amazing storyline . Bella Swan moves to Forks and encounters Edward Cullen, a gorgeous boy with a secret. Are you interested in some other movies made by this director ?

24: U: No, what is its genre?

25: C: It is a drama. Are you also interested in who played in Twilight ?

26: U: Yes, tell me.

27: C: Matt Bushell played in this movie . Would you be also interested in some other movies with this actor ?

28: U: No, tell me rating of the film.

29: C: Rating of the film Twilight is 5.2. Are you interested in the price of Twilight on Amazon ?

We evaluate the last dialog only positively as there are no misunderstandings and the chatbot was able to answer all the user requests correctly.

### 7.3.5 Dialog 5

1: C: Hello there, I am your chatbot for movies. What would you like to know?  
2: U: Hi, what can you tell me about Twilight?  
3: C: Twilight was made by Catherine Hardwicke in the year 2008 and has an interesting storyline . Bella Swan moves to Forks and encounters Edward Cullen, a gorgeous boy with a secret. Would you like to know the price of DVD with the movie called Twilight ?  
4: U: no, who acted in it?  
5: C: For example Sarah Clarke appeared in this film . Would you be also interested in how many people have rated Twilight ?  
6: U: no, what is its genre?  
7: C: That is a drama. Would you be also interested in the screenwriter of Twilight ?  
8: U: Home Alone, tell me something about it.  
9: C: I am sorry, we did not find the required data in our system. Would you be also interested in the genre of Home Alone tell me something ?  
10: U: No, i want to know who acted in Home Alone.  
11: C: I am sorry, we did not find the required data in our system. Would you be interested in its storyline ?  
12: U: Do you know in which movies did Jack Nicholson act?  
13: C: For instance in the film The Evening Star. Are you interested in the language of the film called The Evening Star ?  
14: U: In which year was this film released?  
15: C: The Evening Star was shot in 1996. Are you also interested in who directed The Evening Star ?  
16: U: yes  
17: C: It was Robert Harling's work. Would you like to know who played in this movie ?  
18: U: no, what is its genre?  
19: C: The movie called The Evening Star is a comedy. Would you also like to know the rating of The Evening Star ?  
20: U: no, what is the rating of Catch me if you can?  
21: C: Rating of Catch me if you can is 8.1. Would you be interested in when the film called Catch me if you can was created ?  
22: U: no, who acted in that film?  
23: C: For instance Christopher Walken starred in it . Would you be also interested in some other movies with this actor ?  
24: U: no, whe else acted in it?  
25: C: Christopher Walken starred for instance in the movie called The Dead Zone. Are you interested in when was this actor born?  
26: U: no, would you recommend this movie?  
27: C: Rating of the movie called The Dead Zone is 7.2. Would you be interested



in how many people have rated this movie ?  
28: U: sure  
29: C: The film has 53155 ratings. Are you interested in who made this movie ?  
30: U: no, when was the movie filmed?  
31: C: I am sorry, we did not find the required data in our system. Are you interested in who made filmed ?  
32: U: What do you know about 2012?  
33: C: I am sorry, we did not find the required data in our system. Are you also interested in how many people have rated this movie ?  
34: U: what do you know about 2012?  
35: C: 2012 was directed by Roland Emmerich in the year 2009 and has an engaging storyline . A frustrated writer struggles to keep his family alive when a series of global catastrophes threatens to annihilate mankind. Would you be also interested in some other movies made by Roland Emmerich ?

For conducting this dialog, Stanford NER was used in information extraction module. Similarly, as before there were misclassifications. On line 8 the chatbot did not correctly recognize the movie title and therefore was not able to find it in a database. In the following turn on line 10, a similar mistake happened – NER tagged "i want to know Home Alone" as a movie. The dialog then continued correctly until line 24 where the intent was misclassified. Eventually, there were two additional misclassifications of NER – on line 30 the recognized movie title was "filmed" and on line 32 the detected movie title was "What do you know 2012".

### 7.3.6 Summary

To sum up our experience based on the shown dialogs, we conclude that there are no issues with the provided data or a choice of offered attributes and also the language of the chatbot is proper. Also, the misclassifications of intent and NER are not frequent and, moreover, the chatbot is able to recover from them.

The RNN used for intent classification had an overall accuracy of 0.96 in the shown dialogs which correspond to the results shown previously. On the other hand, the Stanford NER accuracy on the dialogs seems to be comparable to trie method.

# Chapter 8

## Conclusion

In this thesis, we have introduced the topic of chatbots and virtual assistants. We classified chatbots concerning their functionalities and purposes and also commented on how ontologies can be utilized to help chatbots in driving a conversation through a domain.

We followed with a broad description of knowledge organization and why is it necessary. We have also explained the concept of ontologies and their primary purpose to structure information. Then, we took a look at RDF databases and their ability to store a large amount of data in a structured form. And finally, we described the tool for retrieving structured data from such databases.

After that, we listed important tasks of NLP that we divided into two parts – first, natural language understanding, where we described intent classification, named entity recognition and anaphora resolution. Second, we listed the essential properties of natural language generation.

The last chapter that deals with related work begin with the problem of machine learning and evaluation of models in general which is followed by the description of the well-known neural networks with detailed information on frequently used LSTM cells in NLP. We also describe the reasons why the usage of word embeddings is crucial and list possible approaches for their creation. Throughout these chapters, we have overviewed the state-of-the-art conversational AI technologies.

Then, we describe our semi-automatic design of the ontology-based chatbot that uses multiple data sources to provide information to the user, including DBpedia and Wikidata. The whole chatbot is divided into six individual modules that serve different purposes so that the overall architecture works well. Due to the decomposition of the implementation, it allows modularity in the sense of changing a particular implementation. We also use a scraper to extract semi-structured data from a web page. The chatbot does not only use the nomenclature given by Schema.org ontology but it also actively uses it in offering related information to what the user queried before. As a by-product, it learns which attributes are important for the user for next

rounds of conversation. We have also integrated this chatbot into Amazon Alexa and Slack, and additionally discussed how it could easily be done in general with the simple API of our chatbot.

In the previous chapter, we have experimented with two methods of NER that we wanted to compare and choose from. Our proposed approach performed reasonably on personal names based on the data that were available. On the other entities, Stanford NER performed significantly better. We have also experimented with the design and hyper-parameters of the RNN that we use for intent classification, which yielded good results. Eventually, we have tested our application with multiple users by performing short dialogs.

Throughout the whole implementation, we have battled the issue of lack of labeled data. We believe that with more comprehensive labeled datasets the results and capabilities would get improved.

# Bibliography

- M. R. Alam, M. B. I. Reaz, and M. A. M. Ali. A review of smart homes—past, present, and future. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1190–1203, 2012.
- D. Altinok. An ontology-based dialogue management system for banking and finance dialogue systems. *CoRR*, abs/1804.04838, 2018. URL <http://arxiv.org/abs/1804.04838>.
- Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.
- J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.
- J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.
- R. Bhattacharyya and S. M. Hazarika. Intent recognition in a generalized framework for collaboration. *Procedia Computer Science*, 84:123–126, 2016.
- P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.
- A. Bordes and J. Weston. Learning end-to-end goal-oriented dialog. *CoRR*, abs/1605.07683, 2016. URL <http://arxiv.org/abs/1605.07683>.
- D. Britz. Deep learning for chatbots, 2016.
- C.-Y. Chen, D. Yu, W. Wen, Y. M. Yang, J. Zhang, M. Zhou, K. Jesse, A. Chau, A. Bhowmick, S. Iyer, et al. Gunrock: Building a human-like social bot by leveraging large scale real user data.
- D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.
- M. Collins. *Probabilistic Context-Free Grammars (PCFGs)*. 2011. URL <http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/pcfgs.pdf>.

- K. V. Deemter, M. Theune, and E. Krahmer. Real versus template-based natural language generation: A false opposition? *Computational Linguistics*, 31(1):15–24, 2005.
- R. Dey and F. M. Salem. Gate-variants of gated recurrent unit (GRU) neural networks. *CoRR*, abs/1701.05923, 2017. URL <http://arxiv.org/abs/1701.05923>.
- P. Dhillon, D. P. Foster, and L. H. Ungar. Multi-view learning of word embeddings via cca. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 199–207. Curran Associates, Inc., 2011. URL <http://papers.nips.cc/paper/4193-multi-view-learning-of-word-embeddings-via-cca.pdf>.
- I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015. URL <http://arxiv.org/abs/1503.04069>.
- T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.
- S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.
- HPI, 2017. URL <https://hpi.de/naumann/projects/repeatability/datasets/movie-dataset.html>.
- M. Hutson. When will alexa, google assistant, and other ‘chatbots’ finally talk to us like real people? 2017.
- V. Ilievski. Building advanced dialogue managers for goal-oriented dialogue systems. *arXiv preprint arXiv:1806.00780*, 2018.
- D. Jurafsky and J. H. Martin. *Speech and language processing*, volume 3. Pearson London, 2014.
- J. Kim, H.-G. Lee, H. Kim, Y. Lee, and Y.-G. Kim. Two-step training and mixed encoding-decoding for implementing a generative chatbot with a small dialogue corpus. In *Proceedings of the Workshop on Intelligent Interactive Systems and Language Generation (2IS&NLG)*, pages 31–35, 2018.
- G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 971–980, 2017.
- J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

- H. J. Levesque. *Common Sense, the Turing Test, and the Quest for Real AI: Reflections on Natural and Artificial Intelligence*. MIT Press, 2017.
- F. Manola, E. Miller, and B. McBride. Rdf 1.1 primer. *W3C Working Group Note, February, 25, 2014*.
- B. Melichar, J. Holub, and T. Polcar. *Text Searching Algorithms*. 2005. URL [https://cw.fel.cvut.cz/old/\\_media/courses/a4m33pal/melichar-tsa-lectures-1.pdf](https://cw.fel.cvut.cz/old/_media/courses/a4m33pal/melichar-tsa-lectures-1.pdf).
- A. Mikheev, M. Moens, and C. Grover. Named entity recognition without gazetteers. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 1–8. Association for Computational Linguistics, 1999.
- T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.
- T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013b.
- G. E. Modoni, M. Sacco, and W. Terkaj. A survey of rdf store solutions. In *Engineering, Technology and Innovation (ICE), 2014 International ICE Conference on*, pages 1–7. IEEE, 2014.
- M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.
- MovieKeywords, 2018. URL <https://www.the-numbers.com/movies/keywords>.
- V. Nemcik. *Anaphora resolution*. PhD thesis, Master’s thesis, Faculty of Informatics, Masaryk University, 2006.
- J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- J. Pichl, C. FEE, P. Marek, J. Konrád, M. Matulík, J. Šedivý, and C. CIIRC. Alquist 2.0: Alexa prize socialbot based on sub-dialogue models.
- P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL <http://arxiv.org/abs/1710.05941>.
- E. Reiter and R. Dale. *Building natural language generation systems*. Cambridge university press, 2000.
- I. V. Serban, R. Lowe, P. Henderson, L. Charlin, and J. Pineau. A survey of available corpora for building data-driven dialogue systems. *arXiv preprint arXiv:1512.05742*, 2015.

- C. Sutton and A. McCallum. *An introduction to conditional random fields for relational learning*, volume 2. Introduction to statistical relational learning. MIT Press, 2006.
- C. Sutton, A. McCallum, et al. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4(4):267–373, 2012.
- A. Tammewar, M. Pamecha, C. Jain, A. Nagvenkar, and K. Modi. Production ready chatbots: Generate if not retrieve. *arXiv preprint arXiv:1711.09684*, 2017.
- A. Venkatesh, C. Khatri, A. Ram, F. Guo, R. Gabriel, A. Nagar, R. Prasad, M. Cheng, B. Hedayatnia, A. Metallinou, et al. On evaluating and comparing conversational agents. *arXiv preprint arXiv:1801.03625*, 2018.
- J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448. Springer, 2005.
- D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.
- C. Welty, D. L. McGuinness, and M. K. Smith. Owl web ontology language guide. *W3C recommendation, W3C (February 2004) <http://www.w3.org/TR/2004/REC-owl-guide-20040210>*, 2004.
- T. Wen, M. Gasic, N. Mrksic, L. M. Rojas-Barahona, P. Su, S. Ultes, D. Vandyke, and S. J. Young. A network-based end-to-end trainable task-oriented dialogue system. *CoRR*, abs/1604.04562, 2016. URL <http://arxiv.org/abs/1604.04562>.
- T.-H. Wen, M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.
- R. V. Yampolskiy. Turing test as a defining feature of ai-completeness. In *Artificial intelligence, evolutionary computing and metaheuristics*, pages 3–17. Springer, 2013.
- Z. Yan, N. Duan, P. Chen, M. Zhou, J. Zhou, and Z. Li. Building task-oriented dialogue systems for online shopping. In *AAAI*, pages 4618–4626, 2017.
- S. Young, M. Gašić, B. Thomson, and J. D. Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

# Appendix A

## Attached Files

Attached files contain source codes of the movie chatbot, source files of the  $\LaTeX$  file and the master thesis in PDF format. The structure of the directories is described in the following table.

<b>Directory/File</b>	<b>Description/Content</b>
<code>/code</code>	source codes of the movie chatbot in Python
<code>/latex</code>	$\LaTeX$ source files
<code>thesis.pdf</code>	master thesis in PDF format

Table A.1: Description of attached files