Querying NoSQL with Deep Learning to Answer Natural Language Questions

Master Thesis

by

Sebastian Blank

Industrial Engineering and Management M.Sc.

Matriculation Number: 1664468

Institute of Applied Informatics and

Formal Description Methods (AIFB)

KIT Department of Economics and Management

Advisor:	Prof. Dr. York Sure-Vetter
Second Advisor:	Prof. Dr. Andreas Oberweis
External Advisors:	Dr. Florian Wilhelm (inovex GmbH)
	Hans-Peter Zorn (inovex GmbH)
Submitted:	June 27, 2018

Abstract

A challenging task in natural language processing is the development of end-to-end dialog systems that incorporate external knowledge which is stored in databases. This thesis implements a state-of-the-art sequence-to-sequence model that is trained on the Movie Dialog Dataset [Dod+15]. The dataset provides a knowledge base that contains meta data about more than 17k movies, which this thesis stores in an Elasticsearch instance.

Since database operations are non-differentiable, two different ways of training are investigated. On the one hand, this thesis extends the dataset by intermediate labels, which represent the query ground truth, and uses them to train the model before the execution of the database operation. The annotation with intermediate labels required human interaction. On the other hand, policy-based reinforcement learning is used to train the model on the original question-answer pairs.

This thesis finds that training with intermediate labels achieves an executional accuracy of 90.6% and thereby approaches the QA benchmark reported by Dodge et al. [Dod+15]. Training with policy gradient achieves an executional accuracy of 84.2% and performs on a competitive level with an ensemble of Memory Networks. Furthermore, the proposed sequence-to-sequence model generalizes on unknown question patterns.

Zusammenfassung

Eine aktuelle Forschungsfrage im Natural Language Processing ist die Entwicklung von Dialogsystemen, die mit Datenbanken interagieren. Diese Masterarbeit implementiert ein Model nach dem aktuellen Stand der Technik und trainiert es auf dem Movie Dialog Dataset [Dod+15]. Dabei wird die zum Datensatz gehörende Wissensbasis, mit Metadaten zu mehr als 17T Filmen, in einer Elasticsearch Instanz abgelegt.

Aufgrund der Tatsache, dass Datenbankabfragen nicht differenzierbar sind, betrachtet diese Arbeit zwei Ansätze um das Modell zu trainieren. Auf der einen Seite wird das Modell vor der eigentlichen Ausführung der Abfrage trainiert. Zu diesem Zweck wird der Movie Dialog Datensatz um die Abfrage-Grundwahrheit ergänzt. Auf der anderen Seite wird eine Methode aus dem Reinforcement Learning verwendet um das Model auf den ursprünglichen Frage-Antwort Paaren zu trainieren.

Das auf den annotierten Daten trainierte Model kann für 90.6% aller Fragen die korrekte Antwort aus der Wissensbasis abfragen. Damit erzielt es ähnliche Ergebnisse wie das von Dodge et al. [Dod+15] berichtete Referenzsystem. Der Reinforcement Learning Ansatz findet für 84.2% aller Fragen die korrekte Antwort und erreicht damit ein vergleichbares Niveau zu einem Ensemble von Memory Netzwerken. Außerdem konnte festgestellt werden, dass das Model über unbekannte Fragemuster generalisiert.

Acknowledgements

First and foremost, I would like to thank inovex GmbH for the opportunity to write my master thesis within such a supportive and motivating environment. I want to express my deep gratitude to Dr. Florian Wilhelm and Hans-Peter Zorn for their enthusiastic guidance and fruitful discussions.

Furthermore, I would like to thank Prof. Dr. Sure-Vetter, Dr. Achim Rettinger, Boulos El-Asmar and Aditya Mogadala from the Institute of Applied Informatics and Formal Description Methods at the Karlsruhe Institute of Technology (KIT) for their supervision and helpful feedback.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

ANN Artificial Neural Network

API Application Program Interface

ASR Automated Speech Recognition

AT&T American Telephone and Telegraph Company

BGD Batch Gradient Descent

BPTT Backpropagation Through Time

CE Cross Entropy Loss

CNN Convolutional Neural Network

DL Deep Learning

DRL Deep Reinforcement Learning

DST Dialog State Tracker

ELU Exponential Linear Unit

EOS End-of-Sequence Symbol

GloVe Global Vectors (Embedding)

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

IBM International Business Machines Corporation

KB Knowledge Base

LSTM Long Short-Term Memory Network

MBDG Mini-Batch Gradient Descent

MBS Mini-Batch Size

MC Monte Carlo

MDP Markov Decision Process

MemNN Memory Network

ML Machine Learning

MLP Multi Layer Perceptron

MovieDD Movie Dialog Dataset

NLG Natural Language Generation

NLI Natural Language Interface

NLU Natural Language Understanding

NoSQL Not Only SQL

OMDb Open Movie Database

QA Question Answering

QE Query Entity

QF Query Field

ReLU Rectified Linear Unit

RF Response Field

RL Reinforcement Learning

RNN Recurrent Neural Network

SARSA State-Action-Reward-State-Action Algorithm

Seq2Seq Sequence-to-Sequence

SGD Stochastic Gradient Descent

SOS Start-of-Sequence Symbol

SQL Structured Query Language

TD Temporal Difference

VDA Virtual Digital Assistant

List of Symbols

Deep Learning

s	Discrete time step (encoder)
S	Final time step of the episode including time step s
t	Discrete time step (decoder)
T	Final time step of the episode including time step t
x	Input with n elements
x_i	i-th element of input x
$\hat{y}$	Output
y	Ground truth
b, v	Biases
w, W	Weights
w_{ij}	Weight connecting input x_i and neuron j
θ	Parameterization: $\theta = (W, b)$
net_j	Net input of neuron j
h_j	Activation value of neuron j in layer l
L	Depth of a neural network
$n^{(l)}$	Width of layer l
$b^{(l)}$	Bias of layer l
$W^{(l)}$	Weights of layer l
$h^{(l)}$	Hidden state of layer l, consists of activation values of all neurons in l
$\varphi(\cdot)$	Activation function
$\sigma(\cdot)$	Sigmoid activation
$\tanh(\cdot)$	Tangent hyperbolicus
$ReLU(\cdot)$	Rectified Linear Unit

$ELU(\cdot)$ Exponential Linear Unit

$softmax(\cdot)$ Softmax function

$J(\theta)$ Cost function given the parameterization θ

$\nabla_\theta J(\theta)$ Gradient of the cost function with respect to θ

$\frac{\partial J(\theta)}{\partial \theta_1}$ Partial derivative of the cost function with respect to θ_1

$(x^{(i)}, y^{(i)})$ Training sample i that consists of an input x and the corresponding ground turth y

α Learning rate

x_t Input at time step t (if x is a sequence)

$\hat{y}_t$ Output at time step t

h_t Hidden state at time step t

c_t LSTM cell state at time step t

$\tilde{c}_t$ New cell state at time t

f_t Forget factor in LSTM cells at time t

i_t Input factor in LSTM cells at time t

o_t Output factor in LSTM cells at time t

$LSTM(x_t, h_{t-1})$ LSTM that processes input x_t and the previous hidden state h_{t-1}

$h_t^{(l)}$ Hidden state at time step t in layer l

$\overrightarrow{h}_t^{(l)}$ Hidden layer forward pass at time t in layer l

$\overleftarrow{h}_t^{(l)}$ Hidden layer backward pass at time t in layer l

Reinforcement Learning

s, s' states

a action

$\mathcal{S}$ Set of states

$\mathcal{A}$ Set of actions

$\mathcal{S}$ Set of states

$\mathcal{P}$ State transition probability distribution

γ Discount factor

$\mathcal{M}$ Markov decision process which is defined by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$

t Discrete time step

T Final time step of the episode including time step t

A_t Action at time step t

S_t State at time step t

R_t Reward at time step t

τ Trajectory of state-action-reward pairs

π Policy

$\pi(a|s)$ Probability of taking action a in state s under stochastic policy π

π_θ Policy corresponding to the parameterization θ

$\pi(a|s,\theta)$ Probability of taking action a in state s given the parameterization θ

G_t Return following after time step t

$v_\pi(s)$ Value of state s under policy π

$v^*(s)$ Value of state s under the optimal policy

$q_\pi(a,s)$ Value of taking action a in state s under policy π

$q^*(a,s)$ Value of taking action a in state s under the optimal policy

V Estimate of the state-value function v_π

Q Estimate of the action-value function q_π

$h(s,a,\theta)$ Numerical preference for selecting action a in state s based on θ

$b(S_t)$ Baseline function

State-of-the-Art

$\mathcal{V}$ Vocabulary

X_{ij} Co-occurence matrix of word i in context of word j

$h_s, \bar{h_s}$ Hidden states at encoder time step s

h_t Hidden state at decoder time step t

$score(h_t, \bar{h_s})$ Attention score between decoder at time t and encoder at time s

α_{ts} Attention weight between decoder at time t and all encoder time s

α_t Attention weight between decoder at time t and all encoder time steps

c_t Attention context vector at time t

$\tilde{h}_t$ Attentional hidden state at time t

C_t Output at time step t that corresponds to an index of the input vocabulary

$\mathcal{T}$ Tabular knowledge base

Approach

x Sequential input that consists of S elements

x^v Subset of the input that contains the command tokens

x^c Subset of the input that contains the database categories

x^q Subset of the input that contains the question

$emb(x_s)$ Embedding of the input at time s

$\hat{y}$ Output sequence of length T

y Query ground truth sequence of length T

$query(\hat{y})$ Query interface that is filled with the output sequence $\hat{y}$

ω Database response as a sorted or unsorted list with Ω elements

ψ Ground truth answer to the question x^q

e Entity that represents the object of question x^q

e_1 First word of the entity

e_{-1} Final word of the entity

R Reward function

z Positive reward given for valid queries that yield correct results

R^+ Count-based exploration bonus

$count(y_t)$ Count function determines how often y_t was chosen in a mini-batch

T^{R^+} Set of relevant time steps for the exploration bonus

Results and Discussion

N Number of examples in the dataset

N_{vq} Number of valid queries

Acc_{vq} Valid query accuracy

N_{ex} Number of executed queries that yielded a correct result

Acc_{ex} Executional accuracy

N_{qm} Number of examples with a query string match

Acc_{qm} Query match accuracy

N_{qm}^{slot} Number of Examples with a query match in one of the slots (QF, QE or RF)

Acc_{qm}^{slot} Query match accuracy of one of the slots (QF, QE or RF)

λ_{start} Start index of the query entity (corresponds to position of e_1 in x^q)

λ_{end} Final index of the query entity (corresponds to position of e_{-1} in x^q)

L_j Length of the entity in example j of the dataset

κ Share of invalid entities

$d_{start,j}$ Distance between the start index of the ground truth and the predicted entity of example j

Δ_{start} Quality of the first word of the entity in the mini-batch

$d_{end,j}$ Distance between the final index of the ground truth and the predicted entity of example j

Δ_{end} Quality of the final word of the entity in the mini-batch

c Category

N_c^{slot} Number of times category c was chosen to fill a slot (QF or RF)

p_c^{slot} Probability that the model selects category c to fill a slot (QF or RF)

$H^{slot}(p)$ Entropy in the slot (QF or RF)

1 Introduction

"To be able to ask a question clearly is two-thirds of the way to getting it answered."

— John Ruskin

1.1 Motivation

Technology is one of the key drivers guiding the evolution of language and it affects communication from several sides. For instance it increases the speed of information flow, it provides new interfaces, offers the possibility to store information or might by itself be a topic of communication. Furthermore, technology breaks down barriers which seemed intractable. One development which shows the impact of technology is today's influence of machine translation. While translations formerly required skilled translators, technology enables everyone with an Internet access to gain remarkable insights when confronted with information in a foreign language [YD15]. Another major impact is the advent of smartphones. On the one hand, it altered our way of processing information. In the 21st century, information is ubiquitous as social media allows to share news from everywhere around the world. Technology enables people to participate in events of public interest. Presidential speeches, concerts or sport events are accessible from all over the globe. Same applies during the appearance of a crisis, like natural disasters or acts of war. The combination of technologies that connect people from a technical and social point of view, has changed communication between humans significantly. On the other hand, smartphones revolutionized the way humans interact with machines and devices. Touchscreens already existed before the rise of smartphones. But smartphones brought them into daily life. This trend spread to other mobile devices like tablets and even new generations of laptops.

The next step of evolution in communication between man and machine is happening right now. In 2011 Apple presented the iPhone 4s. In combination with hardware improvements the smartphone introduced a digital personal assistant as a complementary service. Such a system can provide information about your personal calender, stock prices or the weather forecast. Other manufacturers recognized the potential of such assistants and caught up. Almost every smartphone one can purchase today will come with a digital assistant that can receive commands in natural language. The current shift in technology highlights the importance of voice as an interface between human and machines. Over the last years leading tech companies like Amazon and Google developed a new generation of devices.

Their digital assistants, Alexa and Google Home, are fully based on voice commands. These assistants do not have displays any more. So far, their main field of application is to control smart home devices, assistance in simple daily tasks and question answering. It is very likely that their capabilities and usage will increase in the next years.

As these considerations only focused on consumer products it is necessary to add that the advance of digital assistance will also cover industries and enterprises. According to a recently published report about the potential of virtual digital assistants (VDAs), the expected amount of VDAs in enterprises will grow to about 840 million VDAs in 2021. This represents an increase of 440 percent compared to the number of VDAs in 2015. On the consumer side, it is estimated that the amount of VDAs will grow from 390 million in 2015 to 1.8 billion in 2021 [Tra16].

Summarizing these considerations leads to the following three points. Firstly, the current development in technology points to natural language as a key interface between man and machine. At this point it is important to emphasize that natural language includes spoken dialog as well as written instructions. Secondly, technology affects communication in general. It removes traditional barriers and offers new opportunities to communicate. Thirdly, the market for digital assistants already exists. It will grow significantly in the next years.

This thesis aims at the problem field of natural language interfaces (NLI). We are living in times where access to information is crucial. A significant amount of today's information is stored and organized in databases. The ability to access such resources is limited to the ones who have mastered a corresponding query language [ZXS17]. Applying a query language in order to access information falls into the same category of problems as accessing information available in a foreign language. The field of NLI aims for the interaction between humans and computers through natural language. A subset of NLI is the research area of semantic parsing. It focuses on the mapping of natural language into logical forms and structured representations. A database query is such a logical form. Semantic parsers explicitly separate between the step of parsing natural language to a logical form and executing the same against a database. However, there is an increased interest in the application of neural network based solutions to tackle this problem in an end-to-end fashion [Lia16]. This thesis will follow this trend and determine the capabilities of such systems. Therefore, the core of this work is to implement a state-of-the-art model which is able to turn natural language questions into database queries. Typically, such models are trained with question-query pairs, as a database induces learning problems caused by broken differentiability. An approach to overcome this problem will be presented and the proposed model will be trained on question-answer pairs. In terms of practical relevance, it is a desirable goal to achieve an end-to-end character. Otherwise the system would require human interaction or intermediate labels. Both would lead to increased costs and reduce the flexibility of the solution. Taking these considerations into account leads to the conclusion that this thesis and the related approach address all three points mentioned earlier.

1.2 Research Questions and Objective

The main objective of this thesis is to investigate on capabilities of neural network based dialog systems. Therefore, the following research questions are presented:

1. Which models achieve state-of-the-art in common dialog tasks?
2. How do such models perform on larger question answering tasks with database interaction?

1.3 Course of Investigation

This thesis splits into six chapters. After this motivational section, **Chapter 2** will present the theoretical foundations of this work. This includes an introduction into dialog systems and a brief summary of their evolution in **Chapter 2.1**. In this context, the standard pipeline architecture will be presented and the shift towards end-to-end solutions will be explained. Later on, details about the underlying methodology of this thesis will be provided. In order to achieve an end-to-end character the proposed model relies on deep reinforcement learning (DRL). **Chapter 2.2** gives an overview on deep learning (DL), which will cover the fundamental concepts of artificial neural networks (ANN), activation functions and gradient-based learning. Subsequently, the family of recurrent neural networks (RNN) will be introduced. Members of this family are able to process sequential data and are therefore useful in problem settings that deal with dialog. An issue with RNNs is that they fail to capture long-term dependencies due to the vanishing gradient problem. Hence, this thesis will discuss the vanishing gradient problem and present long short-term memory networks (LSTM), which overcome this problem. Eventually, **Chapter 2.3** covers the topic of reinforcement learning (RL). It will start by defining the Markov decision process (MDP) as the formal framework of RL and discuss other relevant concepts like rewards, policies, et cetera. DRL deals with the combination of DL and RL and is distinguished into three subclasses. This classification will be used to categorize the approach of this thesis and introduce the *REINFORCE* algorithm.

Chapter 3 will describe the state-of-the-art in natural language processing and present the corpora of available datasets that deal with dialog. In Chapter **Chapter 3.1** sequence-to-sequence (Seq2Seq) models are introduced. They represent an advanced type of neural architecture that leverages two RNNs in an encoder-decoder design. To enable neural networks to process textual inputs, one typically applies pre-trained word embeddings. Such transform words into a numerical representation, which can be used for computation. Word embeddings will be introduced in **Chapter 3.2**. Since the performance of encoder-decoder models suffers to capture long-term dependencies due to a bottleneck between both components, Bahdanau, Cho and Bengio [BCB14] proposed attention mechanisms for Seq2Seq architectures. They will be discussed in **Chapter 3.3**. As mentioned before, a significant share of mankind's information is stored in databases. **Chapter 3.4** describes two ways to interact with knowledge bases (KB) by either computing a probability distribution over all entries in the database (soft-KB lookup) or the production of database queries (hard-KB lookup). **Chapter 3.5** distinguishes between relational and non-relational databases. Since SQL (structured query language) is the main representative of relational databases, non-relational ones are also known as NoSQL (not only SQL) databases. Due to the fact that Elasticsearch is one of the most important NoSQL databases and it has practical relevance at inovex, this thesis implements an Elasticsearch instance in order to store the KB which is related to the use case. The selection of the use case and the corresponding dataset will be discussed in **Chapter 3.6**. Finally, **Chapter 3.7** will conclude the state of the art with respect to research question one.

Chapter 4 introduces the methodology of this thesis. First, **Chapter 4.1** proposes the general approach that is used to answer natural language questions with facts from an external database. Hence, this thesis proposes an attention augmented Seq2Seq model. Since the database operation breaks the differentiability of the system, two ways of training are discussed. While training with intermediate labels requires human interaction, training with policy gradient sustains the end-to-end character. In **Chapter 4.2**, the Movie Dialog Dataset (MovieDD) is analyzed. In this context, this thesis extends the MovieDD by intermediate labels and introduces a reduced training set with 10k examples. Eventually, **Chapter 4.3** presents details of the use case specific implementation.

Chapter 5 provides an answer to the second research question by presenting the results of the conducted experiments. Firstly, **Chapter 5.1** discusses a set of evaluation metrics. Since this thesis extended the MovieDD by intermediate labels, the quality of those labels will be analyzed in **Chapter 5.2**. Afterwards, **Chapter 5.3** shows the results that were achieved by training the model on these question-query pairs. In **Chapter 5.4** the results obtained while training the model on question-answer pairs will be presented. Finally, **Chapter 5.5** compares the outcome of the conducted experiments to the one reported by Dodge et al. [Dod+15]. While they proposed to perform soft-KB lookup with Memory Networks, this thesis implements a hard-KB lookup. Thereby, this thesis provides a comparison of both types of KB interaction on the same dataset.

Finally, **Chapter 6** will conclude the presented findings and outline directions for further research.

2 Theoretical Foundations

This section deals with the theoretical foundations of this thesis. It starts by presenting dialog systems as a generic concept that is based on interaction through natural language. Afterwards, deep learning and reinforcement learning will be introduced due to their importance for the development of end-to-end dialog systems.

2.1 Dialog Systems

Previously, the potential of digital assistants in the near future was highlighted. This section provides the underlying concepts of such systems. Since digital assistants have many synonyms, e.g. conversational agent, dialog system, chatbot or virtual digital assistant, this section will first of all establish a common base in terms of terminology. Subsequently, the standard pipeline architecture of dialog systems will be presented. This architecture dominated the field of conversational agents over the last decades. Eventually, the evolutionary steps in the domain of dialog systems will be shown. In general, one can distinguish between three generations of dialog systems. Lately, there was a shift away from the pipeline architecture towards neural network based end-to-end approaches which represents the third generation of dialog systems.

2.1.1 Terminology

While industry sticks to the phrase digital assistant, research tends to use more precise terms. The generic concept which describes systems interacting with other agents (mostly humans) through natural language is defined as **dialog system** or **conversational agent**. The communication between such systems and their human counterpart is either based on written text, speech or both [JM17]. So while Apple will introduce you to Siri as *your personal assistant*, researchers will call it a dialog system. The group of dialog systems itself can be divided into two subsets according to their objective.

Goal-driven dialog systems have the purpose of gaining information through conversation in order to complete a specific task. Dialogs involving goal-oriented agents are typically short and domain specific. In general, their performance is related to the degree of task fulfillment [Ser$^+$15]. This group of agents includes smartphone assistants, like Siri or Cortana, as well as smart speakers, like Google Home or Amazon Alexa. They complete tasks like messaging, finding restaurants or giving travel directions [JM17].

Non-goal-driven dialog systems are designed to have extended conversations in open domains. They are also called **chatbots** as they mimic the unstructured behavior of human interaction. Instead of retrieving information their goal is to keep the conversation alive. Hence, the agent faces several challenges. On the one hand, the system needs to keep track of the dialog in order to stay consistent with itself and with the current topic of the conversation. This is important due to the fact that the agent might need to retrieve topic specific information. On the other hand, the dialog system needs to decide about an appropriate response to the users input [Ser+15].

Serban et al. [Ser+15] highlight the potential of commercialization as an important research driver. While goal-oriented dialog systems offer an approach to solve clear problem settings involving dialog, the utility of non-goal-oriented dialog systems is not that obvious. Yan et al. [Yan+17] identified a relevant potential in online shopping. They showed that nearly 80 percent of conversations could be categorized as chit-chat. Still, handling those messages and keeping the dialog alive was closely related to user experience [Che+17; Yan+17].

2.1.2 Standard Architecture for Dialog Systems

The interaction between machines and human agents in conversations is a highly complex problem. The standard architecture for dialog systems tackles this problem by dividing it into a set of subproblems. Each one is solved by an individual component that forwards the result to its successor in the process chain. Interaction with the user happens in real-time. Figure 1 shows the standard architecture.

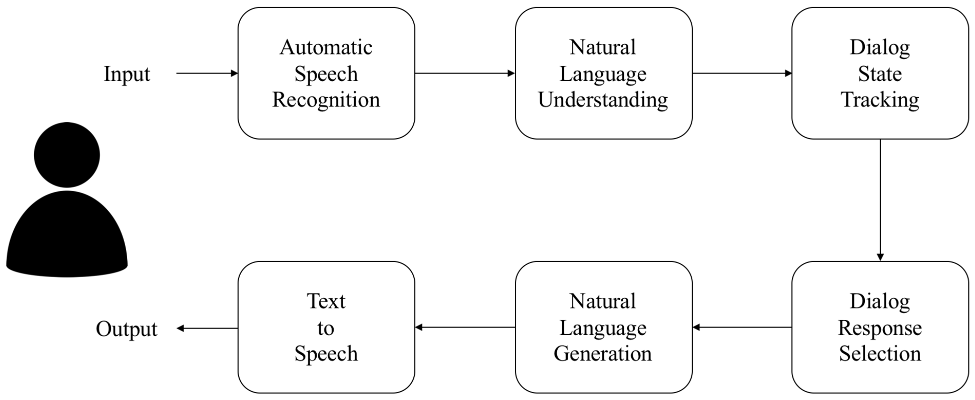

Input
Automatic Speech Recognition
Natural Language Understanding
Dialog State Tracking
Dialog Response Selection
Natural Language Generation
Text to Speech
Output

Figure 1: Standard Pipeline Architecture for Dialog Systems [Ser+15]

The standard architecture is designed as a straight-forward process. Everything starts with a user utterance. In case of spoken dialog systems, the user will input an audio signal. This signal will be transcribed into a textual hypothesis by the **automatic speech recognition** (ASR). Text-based dialog systems only take textual inputs and do not require an ASR. In the next process step, the **natural language understanding** unit (NLU) maps the textual hypothesis into a semantic representation [Ska07]. Table 1 presents an example of a semantic representation for the user utterance: "show restaurants in New York tomorrow". Usually, the semantic representation covers information about the user intent and word-level information, such as named entities. In the example, the semantic representation covers three levels of information. The first level is based on a method called slot-filling. It is applied by different components of dialog systems. The algorithm associates some of the words of the utterance (values) with slots. For example, the slot *destination* is filled with the values *New* and *York*, as they represent the first and last word of the destination. Slot sets can differ depending on the topic. This leads to the second and third level of the semantic representation. They are defined as *intent* and *domain* [Che+17].

Sentence	show	restaurants	in	New	York	tomorrow
Slot	o	o	o	destination	destination	date
Intent	Find Restaurant					
Domain	Order					

Table 1: Example of a Semantic Representation [Che+17]

The NLU outputs the semantic representation to the **dialog state tracker** (DST). At this point, the representation is limited to information contained in the current user utterance. The DST enriches the semantic representation with it's knowledge of the dialog history [Ska07]. If the user utterance, presented in Table 1, succeeded another user input, the semantic representation might be more detailed. Assigning an utterance like: "I love Japanese food" a time step before the current input would lead to a value in the slot *cuisine type*. This is inferred by the DST. The objective of the module is to output a probability distribution over possible dialog states. After the most likely dialog state is determined, the **dialog response selector** derives an action based on the context. This action is still on a semantic level. In the presented example the dialog system could either decide to request additional information, such as the number of people or a city district, or to provide results given the current information. The **natural language generator** (NLG) will transform this semantic representation into an utterance in natural language. In case of text-based dialog systems, the user will receive a written message. Otherwise a **text-to-speech** component will generate an audio output [Ser+15].

2.1.3 Evolution of Dialog Systems

The pipeline architecture of dialog systems has been applied in research and practice. Recent success stories of end-to-end models based on DL challenged the dominating role of the standard architecture. Hence, this chapter provides a brief summary of the developments in the field. For a more detailed overview see [Che+17; JM17; Ser+15].

Literature names ELIZA as one of the first dialog systems in practice [Wei66]. It belongs to the category of non-goal-driven dialog systems. But in contrast to many other chatbots it served a practical purpose. ELIZA was designed to test theories of psychological counseling. A simple rule-based approach simulated the behavior of a psychotherapist by rephrasing statements and posing questions. Later work by Colby [Col81] applied another rule-based approach in a related field. The dialog system PARRY mimicked the behavior of paranoid patients. Both systems belong to the first generation of dialog systems, which was centered around rules and designed by human engineers. Another example from this generation is the conversational agent LUNAR. It provided a natural language interface to a database about moon rocks [Lia16; WKNw72]. This system highlights the fact that even in 1972 engineers saw a potential in accessing databases through natural language, instead of using structured queries. This thesis continues their line of research, but addresses the task with an approach from a succeeding generation of dialog systems.

However, the reliance on human engineers caused the main limitations of rule-based dialog systems. First of all, their development and deployment was time-consuming and expensive. Furthermore, the know-how of individual engineers could turn into a crucial factor for maintenance and debugging. Due to this characteristic the application of rule-based dialog systems was limited to narrow domains [Che+17].

The genesis of the second generation of dialog systems was accompanied by the successful application of statistical methods in several fields of natural language processing. As a result, statistical approaches either replaced or extended existing rule-based solutions. Data was now used to learn statistical parameters in dialog systems. The transition from hand-crafted rules to statistical methods was facilitated by the standard architecture of dialog systems. Due to the pipeline character the complex task of performing a conversation in natural language was divided into simpler subtasks. Especially the NLU and the DST show typical characteristics of a classification problem. While NLU takes user utterances as inputs and maps them to semantic representations, the DST outputs a dialog state based on the semantic representation of the utterance and dialog history. Conditional random fields and hidden Markov models have been successfully applied in both units [Che+17]. Gorin, Riccardi and Wright [GRW97] describe the success story of AT&T. They applied machine learning algorithms to classify free speech problem descriptions in order to allocate operators. Indeed, statistical methods received most of their attention from academic research, whereas commercial applications were dominated by dialog systems from the first generation for a long time. This applies especially for the NLU [Ska07]. Another unit of the standard architecture is more closely related to reinforcement learning than to classification. Literature also refers to the dialog response selector as the dialog policy module. This name emphasizes its relation to reinforcement learning approaches to derive dialog responses. Even today's dialog systems make use of a combination of rule-based agents and reinforcement learning as the dialog response selector to achieve satisfying performance [Che+17].

While the first generation of conversational agents was easy to interpret and debug, the second generation was the opposite. Statistical approaches reduced both, the interpretability and debugability. On the other hand, those methods increased the robustness against noise and ambiguities. But apparently models from the second generation were not powerful enough to scale up to new domains. Hence, they share a major shortcoming of the first generation. Furthermore, it underlines the lack of success in commercial implementations [Che+17].

The success of DL (discussed in chapter 2.2) in other fields attracted researchers to investigate its potential for dialog systems. This represents the starting point of the third generation of dialog systems. In the beginning, research efforts focused on the implementation of deep neural networks in order to replace rule-based or statistical components in the standard architecture. In retrospect, this was a move with an impact on all modules. Hashemi et al. [HAK16] used a DL approach known from image processing in the NLU. They applied Convolutional Neural Networks (CNN) to compute a semantic representation of the input, which was then used to identify the user intent in the NLU. Mrksic et al. [Mrk+16] as well as Henderson, Thomson and Young [HTY13] proposed different types of neural belief trackers. Those kind of models are used in the DST. Besides that, DRL was implemented to predict dialog responses and RNNs showed their potential in the NLG [Che+17].

Another remarkable impact of DL was the effect on research to relinquish of the standard pipeline architecture. This allowed end-to-end dialog system architectures to develop, which show several advantages. First of all, end-to-end dialog systems are able to scale up to new domains. As such, models are only trained on dialog history; there are no underlying assumptions about a domain or dialog state structure [BBW17]. Furthermore, they dissolve the credit assignment problem of the pipeline structure. Considering a standard pipeline system where all components achieve good individual performance results, it is hard to reason about causalities in the connected pipeline. Instead, end-to-end systems are composed as a single unit and optimize a single objective function. By this, they also avoid interdependencies between the different modules which might affect performance [Che+17]. Despite their advantages, end-to-end models have an increased demand for training data, as neural networks are known to be data intensive. Hence, the application of end-to-end dialog systems is limited to problems offering a large amount of conversational data [Lia16].

2.2 Deep Learning

This section introduces deep learning (DL). DL is an important discipline in the field of artificial intelligence (AI). Hence, it will be distinguished from other relevant disciplines in the domain. Afterwards, artificial neural networks (ANNs) will be defined. They represent the fundamental concept behind DL. In general, ANNs are composed of small building blocks called neurons. To gain deeper understanding of ANNs, those blocks will be explained in a first step. Their design is one of the crucial factors to the performance of such networks. Later on, it will be discussed how ANNs learn from experience. Eventually, a group of advanced neural architectures will be introduced.

In the recent past, AI and DL received a lot of media coverage due to use-cases like autonomous driving, where DL is applied to address subtasks like image-based object recognition, e.g. to identify pedestrians or traffic lights; system control, like steering or navigation; or the communication with the driver. Another example which achieved large attention was a DL implementation by Deep Mind [Sil+16]. Their team built an agent that learned how to play Go. Go is a highly complex board game, similar to chess. Their agent was able to beat the world champion in several rounds and thereby continued the line of success, started with Deep Blue in 1996. Deep Blue was a chess computer designed by IBM and defeated the reigning world champion in chess, Garry Kasparov [CHH02].

The success of Deep Blue was one of the big milestones in the emerging field of AI. Figure 2 shows how DL is related to AI and some of its sub-disciplines. Artificial intelligence aims to understand and create machines that think and act intelligent. This definition highlights a distinction between the reasoning process behind an observed behavior (thinking) and the observed behavior itself (acting). This distinction is also known as the hypothesis of weak and strong AI. The **weak AI** hypothesis focuses on the way a machine acts. It is satisfied, if a program acts as if it was intelligent. On the other hand, **strong AI** is concerned about what machines actually think and how they reason [RN03].

Achievements like Deep Blue belong to the group of weak AI. Even though the system outperformed a human benchmark it does not mean that this solution is able to adapt to new problems. The success of weak AI systems is often limited to narrow domains and their application takes place in isolated environments with a set of formal rules. As long as the interaction sticks to the formal framework, such agents seem to act intelligent. But as soon as new patterns arrive or the framework is violated, their behavior becomes inconsistent or even useless. To achieve strong AI it is necessary that systems overcome this problem and gain knowledge about their environment [GBC16]. It is the same factor that counts for humans. A person who is familiar with chess rules is not able to compete with a chess grandmaster. Although they share the same set of rules defining their environment, the grandmaster is superior due to his experience.

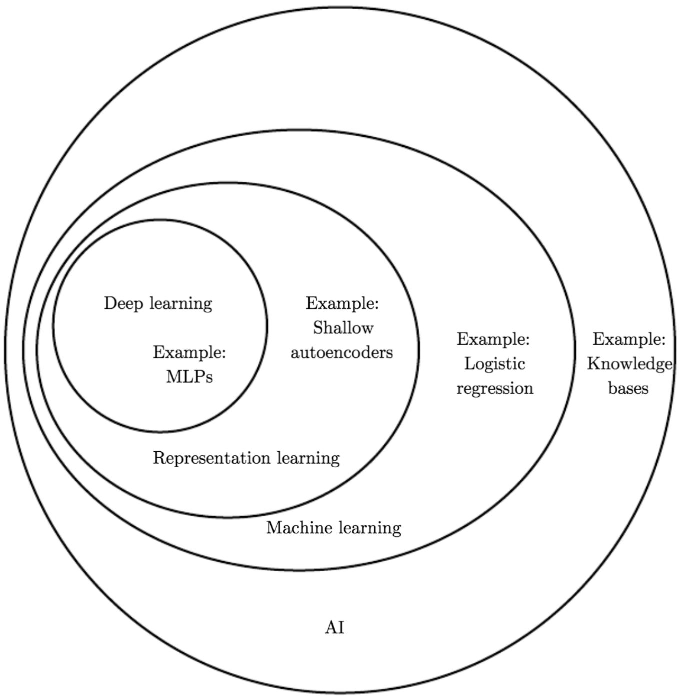


Figure 2: Artificial Intelligence and its Sub-Disciplines [GBC16]

Likewise, **Machine Learning** (ML) integrates the utilization of experience into the field of AI. Mitchell [Mit97] defines, "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." According to this formal definition, ML incorporates three important aspects. A simple example from image recognition will be used to describe these three aspects. Assuming a program receives pictures that show either a dog or a cat, the task T will be defined as a classification of specific images to one of the two categories. Typically, the experience E is provided in form of a dataset where each data point represents an experienced example with a set of features. Given the classification problem each data point corresponds to a processed image, whereas the pixels in the image represent its features. Based on this features, the algorithm will predict class *dog* or class *cat* for each picture in the dataset. As each image is associated with a ground truth, the correctness of the prediction can be evaluated. Hence, the performance measure P will be defined as the accuracy of the predictions in the dataset. Over time, the program will process a lot of images and receive feedback about the correctness of its predictions. Based on this experience, it can adjust its behavior. Typically, ML algorithms are categorized according to their type of feedback. Thereby one can distinguish three categories: supervised, unsupervised and reinforcement learning [GBC16].

In **supervised learning** each example is associated to a label or target. Such algorithms aim to learn a hypothesis that allows them to estimate the labels of a data point [GBC16]. The cat and dog classification is a supervised learning problem as the labels of each image are represented by a string of their ground truth (*dog*, *cat*).

In contrast, **unsupervised learning** algorithms learn patterns in the structure of their data. There is no label or target indicating to which class a data point belongs. Unsupervised learning involves tasks like clustering, which means to group data points according to their similarities. A challenge of unsupervised learning is to define appropriate similarity measures [RN03].

Reinforcement learning is the third category of ML. Like in unsupervised learning there are no associated labels in the data. Instead, RL agents receive feedback in form of a numerical reward signal. In contrast to supervised learning, this signal does not specify how the agent needs to adjust its behavior in order to perform better. So rather than being told what to do, RL systems learn from feedback signals about their environment [RN03; SB17].

The performance of ML algorithms depend on the representation of the data. While traditional methods require pre-processing and feature engineering, there are algorithms that process raw data directly. This group of algorithm is called **representation learning**. As well as traditional methods, such algorithms learn how to map the input data into outputs. Moreover, they learn concepts that implicitly represent the data and improve learning [GBC16].

Deep Learning integrates the idea of representation learning into a hierarchical concept. Such hierarchy starts with low-level representations that identify simple concepts of the raw data. Combining these simple concepts leads to more complex feature representations. As the amount of stacked representations increases, more complex problems can be solved. DL approaches are based on artificial neural networks, which will be described subsequently [LBH15].

2.2.1 Artificial Neural Networks

ANNs mimic the behavior of the human brain based on small building blocks called artificial neurons. First attempts to simplify and formally describe the way human brains work date back to the 1950's and 1960's. The McCulloch-Pitts-Neuron was one of the first works to describe computational signal processing in simplified neurons [MP43]. Later on, Rosenblatt developed another type of artificial neuron, the Perceptron. It was the first of its kind to be implemented [Ros58].

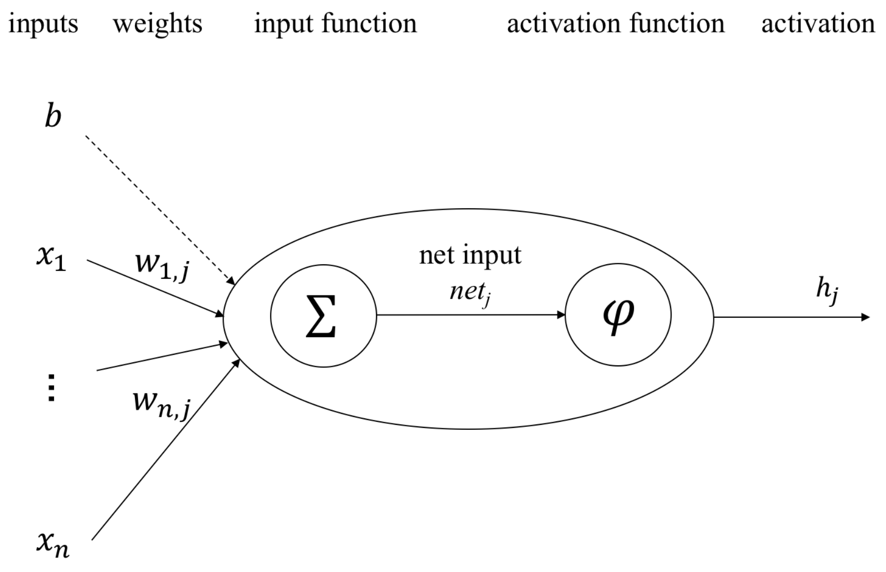


Figure 3: Artificial Neuron [RN03]

Even today's ANNs make use of the basic principle artificial neuron. Figure 3 shows a single neuron j that maps the inputs $x_1, x_2, ..., x_n$ and a bias b into a single activation h_j. The mapping function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is constituted by two operations. The first operation computes the net input net_j of the neuron as

$$net_j = \sum_{i=1}^{n} w_{ij} x_i + b, \tag{2.1}$$

where w_{ij} is a weight vector that connects the input x_i to the neuron j. In a second step the activation value h_j is computed by

$$h_j = f(x, w, b) = \varphi(net_j) = \varphi(\sum_{i=1}^{n} w_{ij} x_i + b), \tag{2.2}$$

where $\varphi(\cdot)$ is an activation function, which will be discussed in Section 2.2.2. In case of deterministic neurons, the activation value h_j equals the signal that is passed to the next neuron. In case of stochastic neurons, the activation h_j can be seen as the probability that the neuron is active [RN03].

The composition of multiple artificial neurons results in powerful network structures. One can distinguish between two different types of network structures. Feedforward networks are acyclic and information flows from inputs to outputs. In contrast, RNNs have cyclic connections and maintain information over time. This is achieved by feeding the output activations back into the neuron and use them as input a time step later [RN03]. RNNs will be discussed in Section 2.2.4, whereas this section will continue and discuss feedforward neural networks.

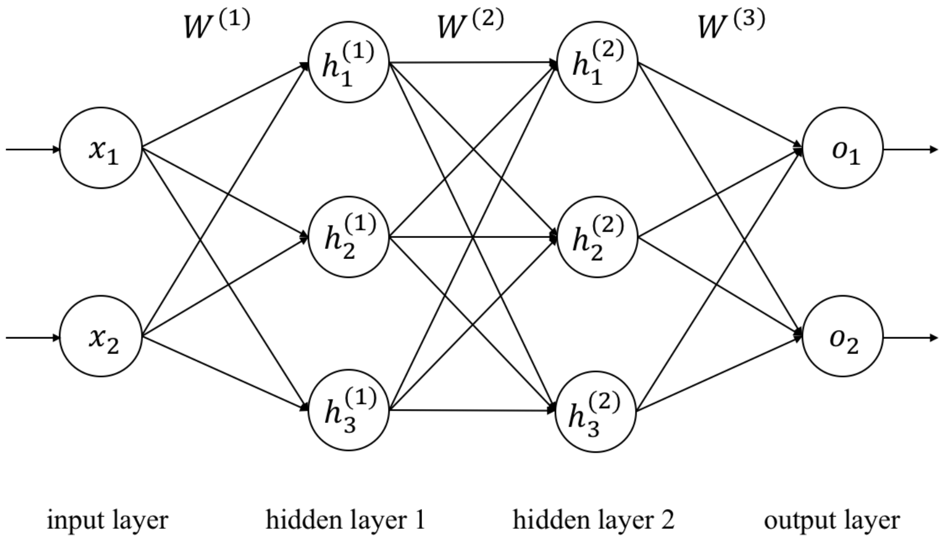


Figure 4: Deep Feedforward Neural Network with two Hidden Layers

The feedforward network depicted in Figure 4 implements the function $f : \mathbb{R}^2 \to \mathbb{R}^2$ to map the input values x_1 and x_2 to the output values o_1 and o_2. The neural network consists of ten neurons which are grouped into four layers and each layer consists of a specific number of neurons. Usually a neuron is connected to all neurons of the previous layer and to all neurons of the subsequent one. If this rule applies for all neurons in the network, it is called fully connected. The first layer of the network is called input layer and it contains the values x_1 and x_2. Incorporating hidden layers into feedforward networks is the key component to their power. According to the universal approximation theorem [Cyb89; HSW89] a feedforward network can approximate any continuous function in $\mathbb{R}^n$ given a single hidden layer with enough units [GBC16]. The depicted network is enlarged by two hidden layers, which contain three hidden units each. While describing a single neuron, the denotation h_j was used to describe the activation value of this neuron j. As the network structure became more complex, the denotation needs to be adjusted. Let $h_j^{(l)}$ be the activation of neuron $j \in 1, ..., n^{(l)}$ in hidden layer $l \in 1, ..., L$. The depth L of the network is defined by its number of hidden layers despite the input layer. In addition, each layer l is characterized by the number of neurons in this layer, also called width $n^{(l)}$ of layer l. For reasons of simplicity the weights connecting two layers will be summarized by a weight matrix $W^{(l)}$ instead of using individual weights on each connection. The dimensionality of the weight matrix is described by $W^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$. Moreover, a bias term $b^{(l)} \in \mathbb{R}^{n^{(l)}}$ is used and added prior to the activation [GBC16; RN03].

As a result one can decompose the mapping function with respect to the chain structure of the network:

$$\begin{aligned} h^{(1)} &= f^{(1)}(x, W^{(1)}, b^{(1)}) \\ h^{(2)} &= f^{(2)}(h^{(1)}, W^{(2)}, b^{(2)}) \\ o &= f^{(3)}(h^{(2)}, W^{(3)}, b^{(3)}). \end{aligned} \tag{2.3}$$

During the introduction of DL, the term representation learning was introduced on a rather abstract level. Artificial neural networks group their neurons layer-wise and each layer performs a transformation of its inputs. The hidden state $h^{(1)}$ is a low-level representation of the input x. Another transformation step in the second layer of the network yields a representation $h^{(2)}$ constructed from the low-level representation $h^{(1)}$. Speaking of a representation hierarchy is nothing else than processing an input through multiple hidden layers. In general, it is hard for humans to interpret representations generated by ANNs. However, analyzing the activations in feature maps of CNNs reveal this feature hierarchy in image processing. Low-level activations correspond to corners and edges, while higher activations capture textures or complex patterns like dog faces or legs [ZF14].

2.2.2 Activation Functions and Softmax

This section will acknowledge the impact of activation functions on the performance of ANNs. Today's state-of-the-art models make use of different nonlinear activation functions. The choice of the activation function influences the performance and capabilities of the model. Figure 5 illustrates four different activation functions.

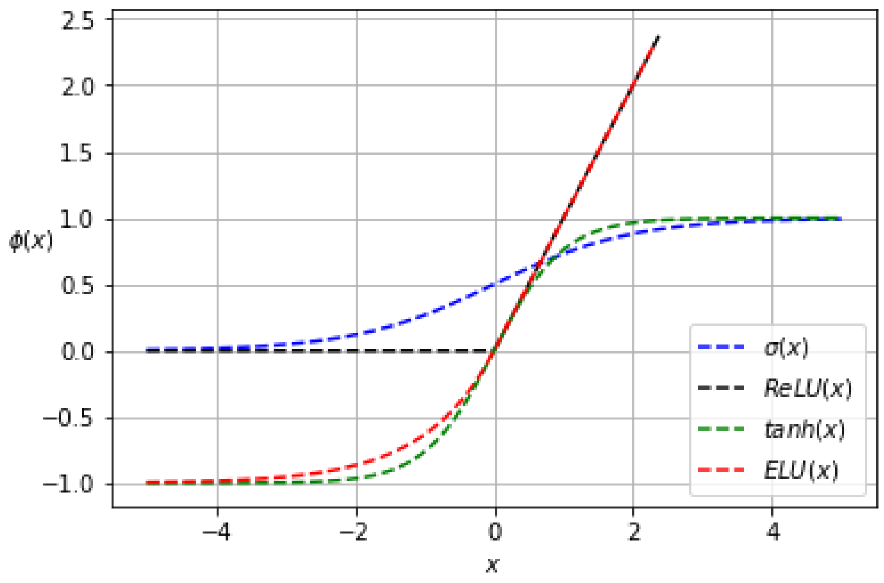


Figure 5: Different Activation Functions

The sigmoid function represents a standard activation and is defined as

$$\varphi(x) = \sigma(x) = \frac{1}{1+\exp(-x)}, \tag{2.4}$$

where $\exp(\cdot)$ is the exponential function. The sigmoid function is continuous and converges to zero if $x \to -\infty$ and to one if $x \to +\infty$ respectively. Furthermore, it is differentiable with an easy derivative. Differentiability is an important characteristic with regards to learning in ANNs. However, the sigmoid function suffers from the vanishing gradient problem [GBC16], which will be described in Section 2.2.6.

The hyperbolic tangent shares the character and shape of the sigmoid function. But instead of converging to zero, one respectively, the hyperbolic tangent approaches -1 if $x \to -\infty$ and one if $x \to +\infty$. Like the sigmoid function it is continuous, differentiable and suffers from the vanishing gradient problem [GBC16]. It is defined as

$$\varphi(x) = \tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}. \tag{2.5}$$

The rectified linear unit (ReLU) was developed to overcome the vanishing gradient problem. It is defined as a piecewise linear function:

$$\varphi(x) = \mathrm{ReLU}(x) = \max(0, x), \tag{2.6}$$

which is easy to optimize and generalizes well. Especially the fixed gradients for $x \geq 0$ and $x < 0$ reduce the computational effort. However, they suffer from a problem called dying ReLU. Large weight updates can cause the input x to be smaller than zero. Due to the fact that ReLUs lack nonzero outputs for $x \leq 0$, their gradient signal will be zero forever and the unit dies. Exponential linear units (ELU) outperform rectified linear units and overcome the dying ReLU problem [CUH15]. They are defined as

$$\phi(x) = \mathrm{ELU}(x) = \begin{cases} \exp(x) - 1 & x < 0 \\ x & x \geq 0 \end{cases}. \tag{2.7}$$

Despite these activations there is another important function for ANNs. Softmax is a function that is able to map discrete variables with n dimensions into the interval between zero and one. Therefore, it is applied to turn the output of a classifier into a probability distribution of n classes, as it sums to one [GBC16]. The softmax function is defined as

$$\mathrm{softmax}(u)_i = \frac{\exp(u_i)}{\sum_j \exp(u_j)}, \tag{2.8}$$

where u is a vector and i, j are indexes.

2.2.3 Learning

This section will discuss how ANNs learn. In general, the learning process of neural networks is divided into three consecutive steps: forward propagation, backward propagation and gradient descent. The sequential execution of those three steps constitutes a training step, also called iteration. Moreover, an iteration over all samples in the training set is defined as an epoch.

2.2.3.1 Cost Function

Subsequently, a cost function, also called error measure or loss, will be defined in order to evaluate the goodness of the approximation of an unknown function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Let $J(\theta)$ denote an arbitrary cost function with respect to

$$\theta = (W, b) = ((W^{(1)}, ..., W^{(L)}), (b^{(1)}, ..., b^{(L)})), \tag{2.9}$$

which represents the parameterization of a neural network with a depth of L. Generally speaking, cost functions are designed to capture the deviation between the ground truth y and the approximation of this ground truth $\hat{y} = f(x, \theta)$. There is a variety of cost functions and while working with ANNs, mean squared error and cross-entropy loss (CE) are the most common ones. This thesis will focus on CE, which is defined as

$$J(\theta) = J(x, y, \theta) = -\sum_i y_i \log \hat{y}_i. \tag{2.10}$$

CE computes the expected negative log-likelihood between the empirical distribution of the training data and the distribution of the model [GBC16].

As defined in Section 2.2, an algorithm learns if it improves a performance measure due to the utilization of experience. Thus, learning in neural networks is an optimization problem under the objective

$$\min_\theta J(x, y, \theta), \tag{2.11}$$

where the cost function $J(x, y, \theta)$ is minimized by adjusting the parameters θ of the network [RN03].

2.2.3.2 Forward Pass

Forward propagation is the first of three steps to train a neural network. Typically, experience for ANNs is provided as a set of tuples: $\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$. The dataset contains m examples and a single tuple consists of a n-dimensional input $x^{(i)}$ and a class label $y^{(i)}$. In this step, the inputs are passed through the different layers of the neural network in order to compute $\hat{y}^{(i)} = o^{(i)} = f(x^{(i)}, \theta)$ [GBC16]. As described in Section 2.2.1 one can resolve this mapping into a composition of functions, which can be executed layer-wise. Figure 6 shows a single neuron which maps the inputs x_1, x_2 to an output $\hat{y} = f(x_1, x_2)$, which is then used to compute the cost $J(x, y, \theta)$.

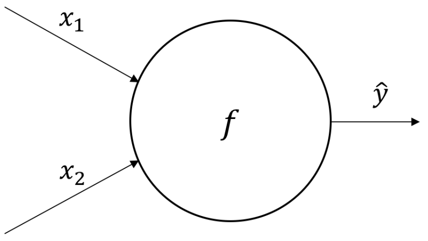


Figure 6: Forward Pass [Cha+17]

2.2.3.3 Backward Pass

Assuming that every hidden unit of the network is partly responsible for the observed cost $J(\theta)$, the weights of the networks are adjusted based on the share of the cost, for which they were responsible. Gradients determine the scale and the direction by which the weights should be adjusted. This leads to the second step in the learning process, which is called backpropagation. Its breakthrough was achieved by a paper in 1986 [RHW86]. Since then, it has been one of the most important algorithms in the domain of neural networks [Nie15].

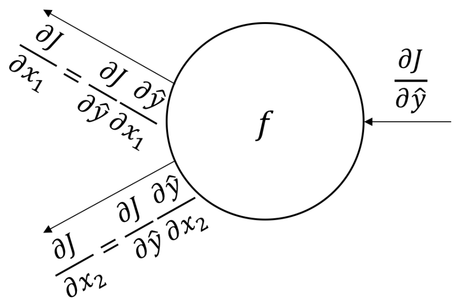

$\frac{\partial J}{\partial x_1} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x_1}$
f
$\frac{\partial J}{\partial \hat{y}}$
$\frac{\partial J}{\partial x_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x_2}$

Figure 7: Backward Pass [Cha+17]

Backpropagation processes the error signal from the output layer, layer-wise through the network until the first layer is reached. First of all, the gradient on the output layer is computed. Figure 7 shows the gradient of the error with respect to the output $\hat{y}$ on the right hand side of the node. Utilizing the chain rule of calculus is the key to propagate the gradient signal through the node. By this, one can rewrite the downstream gradients $\frac{\partial J}{\partial x_1}$ and $\frac{\partial J}{\partial x_2}$ as depicted in the illustration. While the first term, $\frac{\partial J}{\partial \hat{y}}$ is nothing else than the upstream gradient signal, the second term is called local gradient. The local gradient does not consider up- or downstream information. For this reason it is computed and stored during the forward pass. In the backward pass the stored gradient is then used to compute the gradient of the error signal with respect to x_1 and x_2. In case of a network with multiple layers, these gradients will itself work as the upstream signal for the next layer towards the input [Cha+17; GBC16].

The backpropagation terminates as soon as the last gradient in the first layer of the network is computed. As a result, one can formulate the gradient of $J(\theta)$ with respect to all parameters θ in the network:

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}. \tag{2.12}$$

2.2.3.4 Gradient Descent

The final step of a training iteration is gradient descent. In general, gradient descent updates the parameters of the network based on the gradients computed during backpropagation. In practice, there are different implementations of gradient descent. Based on the amount of data they use, one can distinguish between Batch Gradient Descent, Stochastic Gradient Descent and Mini-Batch Gradient Descent.

Firstly, there is Batch Gradient Descent (BGD). It is the standard form of gradient descent and performs the parameter update on the entire dataset. The BGD rule to update the network parameters θ is defined as

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta), \tag{2.13}$$

where α is a learning rate and $\nabla_\theta J(\theta)$ is the gradient of the objective function $J(\theta)$ with respect to the parameters θ. Therefore, one epoch in training only takes one weight update. On the one hand, this is accompanied with slow execution. On the other hand, BGD is guaranteed to converge to global minima for convex error surfaces and to local minima for non-convex error surface [Rud16].

In contrast, the Stochastic Gradient Descent (SGD) uses random samples to update the weights. The SGD update rule is given by

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta, x^{(i)}, y^{(i)}), \tag{2.14}$$

where $x^{(i)}$ refers to a single, random training example i and $y^{(i)}$ refers to the corresponding label. This design allows faster execution and online learning, which means that new examples can be incorporated into the learning process. However, the stochastic character impedes convergence since the weight updates show a high variance. One can counteract this problem with a decreasing learning rate [Rud16].

Eventually, there is the Mini-Batch Gradient Descent (MBGD), which performs its update on a set of examples called mini-batch. Thereby, it reduces the variance of parameter updates and stabilizes convergence. The parameters θ are updated by

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta, x^{(i:i+n)}, y^{(i:i+n)}), \tag{2.15}$$

with $x^{(i:i+n)}$ as a mini-batch of n samples and $y^{(i:i+n)}$ as the corresponding labels [Rud16].

2.2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are an advanced neural architecture. They have been applied to tasks like speech recognition, time series prediction and gesture recognition. All these tasks share that their inputs contain temporal dependencies. This means that the current prediction $\hat{y}_t$ not only depends on the current input x_t, but also on previous inputs $x_{t-1}, ...x_{t-N}$ [LBH15]:

$$\hat{y}_t = f(x_t, x_{t-1}, ..., x_{t-N}, \theta). \tag{2.16}$$

While feedforward networks process their inputs straight through their nodes with no cycles, RNNs do have cyclic connections. This incorporates a memory capability, which is required to keep track of previous inputs. There are two ways to illustrate a RNN, both are depicted in Figure 8. On the left hand side, one can see what is called the *folded* model of the RNN. In general, the model consists of an input x, a hidden state h, which represents the memory, and the output $\hat{y}$. This design allows the RNN to perform the same task for a sequence of inputs. At each time step, the model processes an input x_t and the previous hidden state h_{t-1} to an output $\hat{y}_t$. The weight matrix W_x connects the input to the hidden state. W_h represents the weight matrix which connects the hidden state from the previous time step with the current time step. Furthermore W_y is used to connect the hidden state with the output [GBC16].

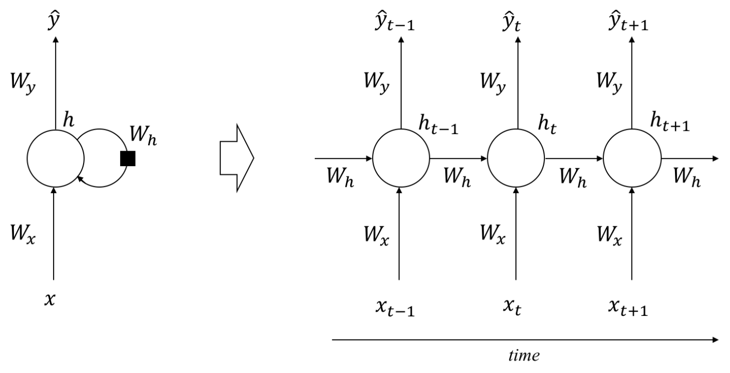


Figure 8: Recurrent Neural Network (Left: Folded - Right: Unfolded into Time) [LBH15]

Since the inputs are represented as a sequence, it is possible to *unfold* the RNN into time. This is depicted on the right hand side of Figure 8. One can see that at time step t, the hidden state h_t depends on two inputs, the event x_t the previous hidden state h_{t-1}. The previous hidden state itself is conditioned on the event x_{t-1} and hidden state h_{t-2}. Furthermore, the unfolded view reveals that the weight matrices are shared over time [GBC16].

Despite the different design, RNNs work in a similar way like feedforward neural network. Assuming that the model is in time step t, it receives the input x_t and the previous hidden state h_{t-1}. The current hidden state h_t is computed by

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b_1), \tag{2.17}$$

where a nonlinear activation is applied on the weighted sum of the inputs x_t, h_{t-1} and the bias b_1 [GBC16]. The output $\hat{y}_t$ can be obtained by applying softmax on a linear transformation of the hidden state h_t:

$$\hat{y}_t = \text{softmax}(W_y h_t + b_2). \tag{2.18}$$

The training procedure of RNNs builds upon the same three steps as the training of feedforward networks. In the forward pass, the RNN computes an output sequence $\hat{y}$ based on the input sequence x and an initial hidden state h_0. Since RNNs share their weights over time, the backward pass is implemented by backpropagation through time (BPTT), which is a generalization of backpropagation[Wer90]. Finally weight updates are performed by gradient descent as discussed in the previous section [GBC16].

However, BPTT can cause the gradients to either grow exponentially or to vanish. This results in unstable learning and the fact that RNNs cannot capture dependencies in longer sequences [Sal+18]. This problem will be discussed in Section 2.2.6.

2.2.5 Long Short-Term Memory Networks

Long Short-Term Memory Networks (LSTM) are designed to avoid the shortcomings of standard RNNs. Hochreiter and Schmidhuber [HS97] proposed the LSTM in 1997. During their path through time their gradients will less likely vanish or explode. This is achieved by connecting weights which work as gates. In particular, those gates enable the LSTM to process information in a long- and a short-term memory [Sal+18]. Since the terminology varies through different resources this section will stick to the state of implementation in Pytorch 0.3.0.

Figure 9 shows the comparison of a RNN and an LSTM unit at time step t. The RNN processes two inputs. On the one hand, the unit takes the information about the event

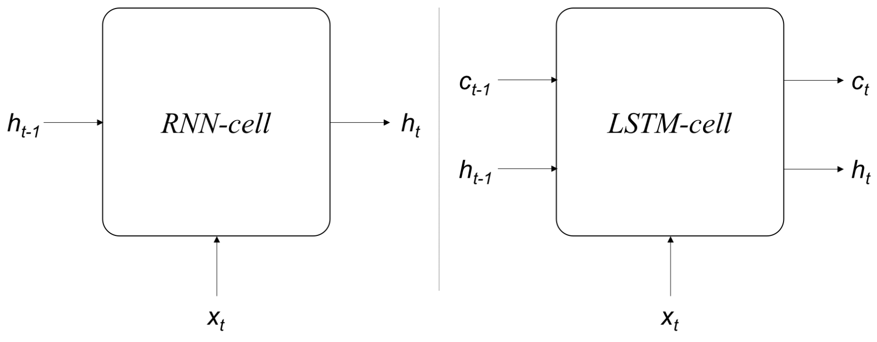


Figure 9: RNN and LSTM

x_t. On the other hand, the hidden state of the previous time step h_{t-1} is fed into the unit. Both inputs are processed through a non-linear transformation resulting in the current hidden state h_t. This one is used to compute the output $\hat{y}_t$. Furthermore, it is forwarded to the unit at $t+1$ in order to update it [GBC16].

In contrast, LSTMs takes three inputs. Like the RNN unit, the LSTM takes the event x_t as an input. But instead of using only one representation of the memory, the LSTM uses two. While the hidden state h_{t-1} represents the short-term memory, the cell state c_{t-1} represents the long-term memory. By processing the inputs through a set of non-linear transformations the LSTM unit updates the cell state c_t and the hidden state h_t. Furthermore, the short-term memory h_t is used to predict the output $\hat{y}$ [GBC16]. Subsequently, this section will discuss how the LSTM processes its inputs.

Generally speaking, an LSTM cell consists of three gates (input, forget and output gate) and the cell state. Each component fulfills a specific task. One can imagine a gate, as a weight which is scaled between zero and one. A closed gate will have a value close to zero and will let no information pass through. However an open gate will let information pass and contain a value close to one [Sal+18].

2.2.5.1 Cell State

The cell state c_t represents the long-term memory. At each time step, a new cell state is computed by

$$c_t = \underbrace{f_t \circ c_{t-1}}_{\text{forget gate}} + \underbrace{i_t \circ \tilde{c}_t}_{\text{input gate}} , \tag{2.19}$$

where $f_t \circ c_{t-1}$ represents the output of the forget gate and $i_t \circ \tilde{c}_t$ refers to the output of the input gate. The forget gate determines, how much of the information of the long-term memory c_{t-1} will stay in the long-term memory in t. This is achieved by the forget factor f_t. If it is close to one, the long-term memory will keep almost everything it stored in $t-1$. If it is close to zero, the LSTM will forget things stored in the long-term memory.

The input gate decides how much new information will find its way into the long-term memory. The input factor i_t works as the gate which decides whether information from a new cell state $\tilde{c}_t$, just based on short-term information, will enter the long-term memory c_t or not. Again, a value near zero means that nearly no information passes the gate. Whereas a value close to one means that most of the information flows through. The cell state is crucial to the performance of the LSTM [Ola15].

2.2.5.2 Forget Gate

During the update of the cell state c_t, the forget gate takes the previous cell-state c_{t-1} and weights it with the forget factor f_t. This forget factor is computed by

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f), \tag{2.20}$$

where the previous hidden state h_{t-1} is multiplied with a weight matrix W_{hf} and the event information x_t is multiplied with another weight matrix W_{xf}. The sum of both products in addition to a bias b_f is then fed into a sigmoid function. Both inputs contain information of short-term relevance. If there is a positive relation with the long-term memory, the LSTM will keep such memories. The matrices W_{xf}, W_{hf} and the bias b_f belong to the forget gate only and are learned during training. The forget gate was introduced as an extension to the first version of LSTM [GSC99].

2.2.5.3 Input Gate

The input gate is responsible to decide which parts of the short-term information are relevant for the future. The short-term information consists of the short-term memory in form of the hidden state h_{t-1} and the current event x_t. Both information are combined into the input factor:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i), \tag{2.21}$$

with b_i as a bias and W_{xi}, W_{hi} as weight matrices that are learned during training. Furthermore, the short-term information is combined to a new cell state $\tilde{c}_t$:

$$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \tag{2.22}$$

where the weight matrices W_{xc}, W_{hc} and the bias b_c are learned during training [GBC16; Ola15].

2.2.5.4 Output Gate

Eventually, this section introduces the output gate, which is responsible to update the hidden state from h_{t-1} to h_t. Therefore, an output factor is computed by

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o), \tag{2.23}$$

where W_{xo}, W_{ho} represent weight matrices and b_o is a bias. These parameters are learned during training. Afterwards, the output factor is used in order to compute the updated hidden state:

$$h_t = o_t \circ \tanh(c_t). \tag{2.24}$$

This means that the output gate decides which informations in the long-term memory are relevant in the near future. The output $\hat{y}_t$ is produced by an additional linear transformation:

$$\hat{y}_t = W_y h_t + b_y, \tag{2.25}$$

where W_y is a weight matrix and b_y is a bias. Both parameters are learned during training. Finally, the unit passes the updated hidden state h_t and cell state c_t to the next time step [GBC16; Ola15]

2.2.5.5 The LSTM Big Picture

LSTMs are an approach that overcomes the vanishing gradient problem. They have shown promising results in various domains like translation, voice recognition [GMH13] or image captioning [Xu$^+$15]. Figure 10 shows an LSTM cell, which implements the different gates and the cell states as discussed before.

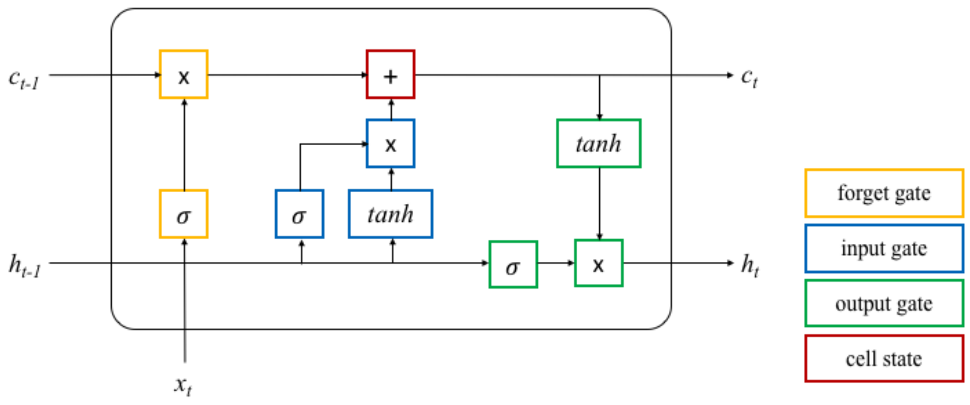


Figure 10: LSTM Unit [Ola15]

Henceforth, this thesis will denote the update of the hidden state by

$$h_t = LSTM(x_t, h_{t-1}). \tag{2.26}$$

This simplified rule represents an abstract summary of the interactions within an LSTM cell. The update of the cell state is implicitly performed according to the previous explanations.

Despite their potential to capture long-term dependencies, LSTMs dramatically increase the computational cost of a model. Since the introduction of LSTMs, there have been attempts to outweigh this disadvantage. The Gated Recurrent Unit (GRU) by Cho [Cho+14] simplifies the LSTM architecture. It reduces the number of gates, by combining the input and the forget gate to a so called update gate. Furthermore, it only uses one representation of the memory, named working memory, which makes GRUs less computational expensive [Cho+14].

Since their introduction, several adjustments have extended the standard LSTM architecture. In this context, Greff provides an analysis of different LSTM variants [Gre+17]. Especially, peephole connections gained a lot of attention. They increase the influence of the cell state by taking it into account while computing the gate factors: f_t, i_t and o_t [GS00].

2.2.5.6 Bidirectional LSTMs

Bidirectional LSTMs build upon the standard LSTM architecture. However, their design realizes the basic idea to present each training sequence $x = x_1, ..., x_T$ forward and backward to the network. This is achieved due to two separate hidden layers which are connected to the same output layer [GS05]. While the **forward layer** processes the inputs from x_1 to x_T, the **backward layer** processes the sequence the other way around: x_T to x_1. As a result, one can observe the forward hidden state $\overrightarrow{h}_t$ and the backward hidden state $\overleftarrow{h}_t$. One can compute the output $\hat{y}_t$ by

$$\hat{y}_t = W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y, \tag{2.27}$$

where $W_{\overrightarrow{h}y}$ and $W_{\overleftarrow{h}y}$ are weight matrices, whereas b_y is a bias [GJM13].

2.2.5.7 Deep LSTMs

Deep LSTM architectures can be created by stacking several LSTM layers on top of each other. Assuming an architecture with L hidden layers, one can generalize the update rule to [GJM13]:

$$h_t^{(l)} = LSTM(h_t^{(l-1)}, h_{t-1}^{(l)}). \tag{2.28}$$

By this adjustment, the hidden layer $h_t^{(l-1)}$ is fed into the LSTM cell at layer l. The original input sequence is defined as $h^{(0)} = x$. Note that for a bidirectional LSTM, all gates and the cell state are defined twice, once for the forward pass and a second time for the backward pass. Furthermore, the output of the network is computed based on the forward and backward hidden state of the last hidden layer L [GJM13]:

$$\hat{y}_t = W_{\overrightarrow{h}y}\overrightarrow{h}_t^{(L)} + W_{\overleftarrow{h}y}\overleftarrow{h}_t^{(L)} + b_y. \tag{2.29}$$

Figure 11 illustrates a deep bidirectional LSTM with two stacked hidden layers. Both hidden layers contain a separate forward and backward layer. At the first hidden layer, the forward pass reads the input sequence from x_1 to x_T, whereas the backward pass reads the sequence the other way around. The forward layer 1 passes its hidden layer state $\overrightarrow{h}_t^{(1)}$ to the forward layer 2. In addition with $\overrightarrow{h}_{t-1}^{(2)}$ it is used to update $\overrightarrow{h}_t^{(2)}$, $\overrightarrow{c}_t^{(2)}$ respectively. For reasons of simplicity, the visualization only shows the hidden layer states. However, during training, all gates and the cell state are computed as discussed before. Furthermore, one can see that the output $\hat{y}_t$ is conditioned on the hidden state of the second hidden layer at time step t: $\overrightarrow{h}_t^{(2)}$ and $\overleftarrow{h}_t^{(2)}$.

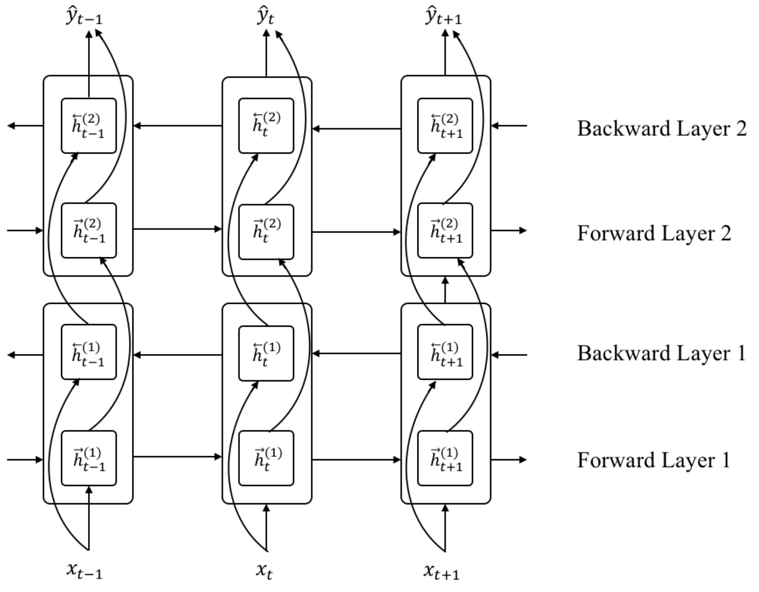

$\hat{y}_{t-1}$
$\hat{y}_t$
$\hat{y}_{t+1}$
$\overleftarrow{h}_{t-1}^{(2)}$
$\overleftarrow{h}_t^{(2)}$
$\overleftarrow{h}_{t+1}^{(2)}$
Backward Layer 2
$\overrightarrow{h}_{t-1}^{(2)}$
$\overrightarrow{h}_t^{(2)}$
$\overrightarrow{h}_{t+1}^{(2)}$
Forward Layer 2
$\overleftarrow{h}_{t-1}^{(1)}$
$\overleftarrow{h}_t^{(1)}$
$\overleftarrow{h}_{t+1}^{(1)}$
Backward Layer 1
$\overrightarrow{h}_{t-1}^{(1)}$
$\overrightarrow{h}_t^{(1)}$
$\overrightarrow{h}_{t+1}^{(1)}$
Forward Layer 1
x_{t-1}
x_t
x_{t+1}

Figure 11: Deep Bidirectional LSTM [GJM13]

2.2.6 Challenges

This section presents challenges for machine learning algorithms in general and RNNs in particular. While the capability to generalize on unseen data is a challenge that many algorithms face, the vanishing and exploding gradient problem is mainly faced by RNNs.

2.2.6.1 Generalization

One of the key challenges in machine learning is to develop a model architecture that generalizes. **Generalization** is the ability of a model to perform well on new, unseen examples. Therefore, one typically divides the available dataset into three separate splits: training set, validation set and test set. While the training set is used to train the model and adjust the parameters, the test set is used to evaluate the final performance of the trained model. The validation set is used to adjust the hyperparameter settings. Figure 12 shows the trade-off between the capacity of the model and the error curves on the training set and the generalization error. In order to achieve a good generalization, one should aim to reduce the training error itself, as well as the gap between training and generalization error. As long as both error functions are decreasing, the model is underfitting. This means, the model is not able to represent the data sufficiently. After the model reached the optimal capacity, the training error decreases further, whereas the generalization error starts to increase. This is called the regime of overfitting. At this point, the model starts to memorize the input data, instead of using an generalized approximation. Techniques which aim to prevent overfitting are called **regularization** methods [GBC16].

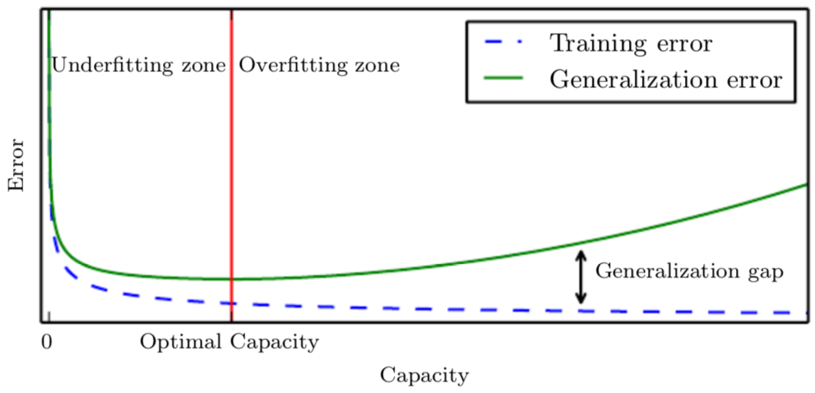


Figure 12: Trade-off between Model Capacity and Error [GBC16]

Dropout is a method which favors regularization in ANNs. It was proposed by Srivastava et al. [Sri+14]. One can imagine dropout as a technique to train a group of sub-networks of the original one. The algorithm mutes non-output neurons with a specified probability. In each iteration another set of neurons will be removed from the network and reactivated after the epoch. Thus, the remaining neurons of the network need to be adjusted in a way that they will capture the information of the "lost" neuron in the future. The network will generalize better over time. Dropout is computationally inexpensive and it works nearly everywhere where gradient descent is applied. A disadvantage of dropout is that it reduces the effective capacity of a model. Therefore the model size needs to be increased [GBC16].

Another common regularization method is **early stopping**. This methodology uses the validation set in order to figure out at which point the generalization error starts to increase while the training error still decreases. In other words, it detects the time when the model starts to overfit. Typically, this is achieved by evaluating the generalization error every n epochs. If the validation error does not decrease during a predefined patience p, the training process is stopped. Early stopping is a popular regularization method due to its simplicity and effectiveness. Further regularization methods are e.g. weight decay, input noise or weight noise [GBC16].

2.2.6.2 Vanishing and Exploding Gradients

The family of recurrent neural architectures suffer from a phenomenon connected to their application of BPTT to update their weights. Literature refers to this problem as **vanishing and exploding gradient** problem [BSF94; Hoc91; Hoc+01].

The **vanishing gradient** problem makes it hard for RNNs to keep track of dependencies for longer sequences. This is caused by the interdependence of two effects. First, the standard nonlinear functions used in RNNs are tanh- and sigmoid-activations. Both activation functions saturate and therefore have gradients close to zero. Second, to learn the weights in the recurrent structure of RNNs, BPTT is applied. BPTT performs multiple multiplications of the gradients, which lets them converge to zero. This means that the magnitude of long-term dependencies is exponentially smaller than the one of short-term dependencies. As a result, it is hard to update the weights, as the direction of parameter adjustments becomes blurred [GBC16; Sal+18]. In section 2.2.5 long short-term memory networks were introduced as an approach that overcomes the vanishing gradient problem.

The **exploding gradient** problem causes the gradients of the network to blow up exponentially and leads to oscillating weights [Hoc+01]. Gradient clipping is an approach which was introduced to avoid this phenomenon. There are various forms of gradient clipping, but in general it can be described as a normlization of the gradient, if the gradient exceeds a certain treshold [Sal+18]. E.g. Mikolov [Mik12] proposed to clip the parameters of a mini-batch element-wise before the parameter. Later on, this idea was extended by Pascanu, Mikolov and Bengio [PMB13], which clipped the norm of the gradient. In theory, their approach had the advantage that it guarantees to maintain an update in the gradient direction. However, in practice both approaches worked similar [GBC16].

2.3 Reinforcement Learning

Reinforcement Learning (RL) is based on the paradigm of trial-and-error learning known from psychology. An agent gains knowledge through interaction with its environment and adjusts its behavior with respect to feedback signals. In general, problems that show the characteristic of sequential decision making can be tackled with RL methods [Aru+17]. Literature provides many examples showing the variability of RL applications through multiple domains. In robotics, for example, RL models are applied to enable machines to learn from human demonstration [Arg+09]. First contributions in this field treated problems such as balancing an inverted pendulum on a cart [MC68]. The implementation of Temporal Difference learning on Backgammon in 1992 was a milestone of RL in gaming [Tes92]. Lately, researchers of DeepMind were able to achieve superhuman performance in Atari video games [Mni+13; Mni+15] and beat the World Champion in Go with a RL model [Sil+16].

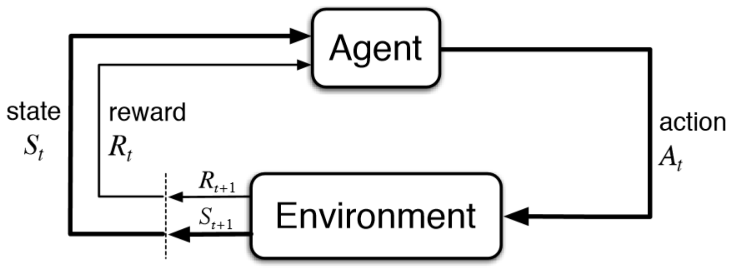


Figure 13: Interaction between an Agent and its Environment [SB17]

The interaction between the agent and its environment is depicted in Figure 13. At each time step $t \in \{0, 1, 2, ..., T\}$ the agent receives a signal about the current state S_t and deduces an action A_t according to some internal strategy, also called policy $\pi(A_t|S_t)$. This action will lead to a new state S_{t+1} of the environment and is rewarded with a numerical feedback R_{t+1}. This feedback signal indicates how good the previously chosen action was. In the next iteration, the agents will choose its action based on the new state S_{t+1}. The interactions between agent and environment result in a sequence of state-action pairs with associated rewards, also called trajectory τ which is defined by [SB17]

$$\tau = (S_0, A_0, R_1), (S_1, A_1, R_2), ..., (S_{T-1}, A_{T-1}, R_T). \qquad (2.30)$$

The following section introduces the Markov decision process (MDP) which is used to give a formal description of the RL environment. Afterwards, further elements of RL are set into context. Subsequently, a distinction between three different categories of RL models will be presented and policy gradient will be introduced as the method applied in this thesis. Finally, some remarks on typical challenges in reinforcement learning will be presented.

2.3.1 Markov Decision Process

Generally speaking, models allow to represent problems in a simplified version due to a set of well defined rules. RL uses Markov Decision Processes to achieve such simplification and describe the environment with the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$:

- $\mathcal{S}$ is a set of states. The initial state s_0 is sampled from $\mathcal{S}$.
- $\mathcal{A}$ is a set of actions.
- $\mathcal{P}$ is a state transition probability distribution, $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$.
- $\mathcal{R}$ is set of reward functions, which maps $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$.
- γ is a discount factor, $\gamma \in [0, 1]$

The key characteristic of MDPs is that they satisfy the Markov Property. According to this property the current state captures all relevant information in the system. Which means their transitions only depend on the current state and not on the past: $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t]$ [SB17].

2.3.2 Further Definitions

During the interaction with the environment, the agent follows the objective to learn a policy which maximizes the expected cumulative rewards. As a result, the agent is able to ignore immediate rewards to achieve higher rewards in the long run. The return G_t at time step t is defined as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{T} \gamma^k R_{t+k+1}, \tag{2.31}$$

where R_t is the reward at time step t and γ is a discount factor. If the discount factor γ is close to 0, the agent acts *myopic* and is more concerned about immediate rewards. If γ is close to 1, the agent will also consider future rewards. In case of episodic tasks it is likely to choose $\gamma = 1$, as such tasks are only rewarded after an episode finished. Alternatively, one can take $T = \infty$ to cover MDPs over infinite sequences. For those tasks it is important that $\gamma < 1$ in order to guarantee convergence.

After the agents objective was defined, this section will continue and discuss the terms policies and value-functions. Both help the agent to achieve its goal and maximize the expected return. In general, a policy defines the way an agent behaves in a certain state. This work will focus on stochastic policies. A stochastic policy maps the current state to a probability distribution over possible actions and than samples an action according to this distribution: $a \sim \pi(s, a) = \mathbb{P}\big[A_t = a|S_t = s\big]$. As the agent gains experience over time it will adjust its policy in order to improve its expected return.

Given the agent is currently in state s, the state-value function v_π is defined as

$$v_\pi(s) = \mathbb{E}_\pi\Big[G_t|S_t = s\Big] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \forall s \in \mathcal{S}, \tag{2.32}$$

where it equals the expected return, assuming the agent will only follow policy π in the future.

The action-value q_π is defined similarly. It assumes the agent in state s, will take action a and follow policy π in the future:

$$q_\pi(s, a) = \mathbb{E}_\pi\Big[G_t|S_t = s, A_t = a\Big] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]. \tag{2.33}$$

As shown in Sutton and Barto [SB17], one can rewrite the state-value or action-value function, which results in a recursive relationship. This relationship describes the relation between the value of a state and the value of the subsequent state. Equation 2.34 shows the derivation of this recursive relationship. The final expression is called Bellman equation for v_π. The same logic applies in order to rewrite the Bellman equation for q_π. Due to its non-linearity and recursive character, there are many approaches that solve the Bellman equation iteratively.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}\Big[G_t|S_t = s\Big] \\ &= \mathbb{E}\Big[R_{t+1} + \gamma G_{t+1}|S_t = s\Big] \\ &= \mathbb{E}\Big[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s\Big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \forall s \in S \end{aligned} \tag{2.34}$$

Since the objective of the agent is to maximize the expected return in a MDP, the optimal value function yields the maximum value over all policies. This applies for both, the optimal state-value $v^*(s)$ and the optimal action-value $q^*(s,a)$:

$$\begin{aligned} v^*(s) &= \max_\pi v_\pi(s) \\ q^*(s,a) &= \max_\pi q_\pi(s,a). \end{aligned} \tag{2.35}$$

Moreover one can define an optimal policy π^*. The expected return of an optimal policy is greater than or at least equal to the expected return of all other policies. An optimal policy satisfies $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$ [SB17].

2.3.3 Three Classes of RL-algorithms

While the previous section introduced the terminology of RL, this section will distinguish between different kind of RL problems in order to introduce the group of algorithms which is relevant to this work.

The pool of RL algorithms offers several possibilities to group them in different ways. A popular way to do so, is illustrated in Figure 14. On the one hand side there are algorithms which are given a state or state-action pair and compute the corresponding value function or an estimate of it. On the other hand there are methods which directly search for a policy. Most likely, those methods rely on a parameterized policy π_θ. Besides those two categories there is a hybrid which combines value functions and policy search. Such models are called actor-critic methods based on the simulated interlude between an actor, which represents the policy, and a critic, which gives feedback by its value function. In contrast to other baseline approaches for policy search, the value function of the critic is actually learned. This fact leads to more powerful models [Aru+17]. Which category one should favor depends on the individual problem settings. Concepts of all three classes have shown their potential in application.

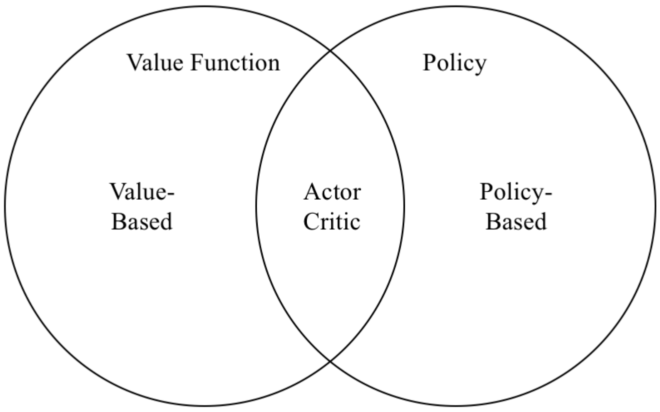


Figure 14: Value Function vs. Policy-Based RL [Sil15]

2.3.3.1 Value-based Methods

This section will provide insights into the class of value-based methods. For this purpose, the concepts of model-based and model-free reinforcement learning will be distinguished. In a model-based environment, an agent can estimate how the environment will change given the agents actions. The model allows him to predict the next state and the next reward given the current state and an action. Settings which involve an model-based method are also called planning problems. In such situations, one can compute an optimal solution before an action is actually taken. The group of model-based algorithms include heuristic search and dynamic program methods, like value iteration or policy iteration [SB17]. Since the latter settings are not part of the primary focus of this study, they will not be discussed further.

In contrast to mode-based methods, model-free approaches have no intuition how the environment will change in response to an action. Such models follow paradigm of trial-and-error learning through interaction with the environment. Common model-free algorithms are Monte Carlo (MC) and Temporal Difference (TD) methods. Both approaches incorporate the capability to learn from experience. Therefore, both methods use an approximation V of v_π, Q of q_π respectively. As soon as the agent gains experience when following policy π both methods will update their estimate based on a target value Y and a step-size parameter α. A generalized form of update rules for state-value and action-value function estimators is provided below [Aru+17]:

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha\Big[Y - V(S_t)\Big] \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[Y - Q(S_t, A_t)\Big]. \end{aligned} \tag{2.36}$$

The main difference between both methods is how they define their target value and how this affects the update of the estimator. Monte Carlo methods rely on the actual return as their target $Y = G_t$. This means that they need to wait until a terminal state is reached and the return is known. Such tasks are also called episodic tasks. For this reason, the experience of MC methods is split up into an episode-to-episode sense. They are not able to learn step-by-step (action-by-action). Such is also called off-policy learning [SB17].

While MC methods need to wait until the end of an episode, TD methods only need to wait until the next step. Instead of updating their estimates based on the actual return, TD methods update their estimates based on other learned estimates. This concept is known as bootstrapping. The state-action-reward-state-action (SARSA) algorithm and Q-learning are methods which actually implement the idea of TD learning. Instead of approximation the state-value function v_π, both algorithms determine Q as an estimate of q_π. SARSA is an on-policy learning algorithm which uses transitions generated in the episode and sets its target $Y = R_t + \gamma Q(S_{t+1}, A_{t+1})$. Q-Learning instead, is an off-policy learner and uses an approximation of q^* as its target: $Y = R_t + \gamma \max_a Q(S_t, a)$ [Aru+17].

In order to learn the value function of a RL problem, the agent needs to keep track of the impact of its actions in a certain state. In smaller MDPs it is sufficient to store the corresponding samples in a lookup table. But as soon as the state-action space becomes larger, this leads to an problem regarding the efficiency and time to convergence [RN03].

Function approximators are an approach to escape this limitation and introduce generalization into RL. Instead of a tabular approach, function approximators provide an estimation based on their parametrization. Over time, their parameters are adjusted in order to match the observed samples [RN03]. During the motivation of RL the example of TD-Gammon, which mastered the game of Backgammon, was mentioned. Indeed, it was an early success of using neural networks for function approximation [Tes92]. Mnih et al. [Mni+15] used Convolutional Neural Networks to approximate the $Q^\pi(s, a)$ of state-action pairs in a video game. The network directly learned from the visual interface of the game. A tabular representation of the space-action space would have had a size of $|\mathcal{S}| \cdot |\mathcal{A}| = 18 \cdot 256^{3 \cdot 210 \cdot 160}$ [Aru+17]. By using a function approximaton $\hat{q}(s, a, \theta) \approx q_\pi(s, a)$, they were able to reduce the complexity to the dimensionality of θ and the effort to update those parameters.

2.3.3.2 Policy-based Methods

So far, algorithms aimed to either compute or estimate the state-value or the action-value of a RL problem. A policy was derived from the value function. However, policy-based methods directly search for a policy instead of taking an intermediate step to determine a value function. That is why policy-based methods are also called policy search algorithms. Typically, this is achieved by a parameterized policy π_θ. The objective of policy-based methods is to maximize the expected return given a parameterization $\mathbb{E}\big[G|\theta\big]$. ANNs are a likely choice to encode a parameterized policy. There have been successful approaches to train them with gradient-free and gradient-based methods.

Gradient-free methods require a search heuristic on a set of predefined class models. Therefore they are mainly applied to low-dimensional parameter spaces [Aru+17]. Koutnik et al. [Kou+13] investigated on the capabilities of evolutionary algorithms instead of using backpropagation. Their work is an example of gradient-free training and relies on a population of RL agents. Such tend to have a natural behavior which favors the exploration of the parameter space. Furthermore, they have the advantage be able to optimize non-differentiable problems [Aru+17].

On the other hand, there are gradient-based methods. Subsequently, the focus will be on gradient-based learning for stochastic policies. Such policies can be written as $\pi(a|s,\theta) = \mathbb{P}\big[A_t = a|S_t = s, \theta_t = \theta\big]$. In a discrete action space one can compute a stochastic policy based on numerical preferences $h(s,a,\theta) \in \mathbb{R}$. The softmax distribution allows to transform the outputs of a neural network into a probability distribution which can represents a stochastic policy:

$$\pi(a|s,\theta) = \frac{\exp(h(s,a,\theta))}{\sum_b \exp(h(s,b,\theta))}, \tag{2.37}$$

where $h(s,a,\theta)$ is a numerical preference. This transformation assigns the highest numerical preference in each state to the highest probability of being selection. By this, the softmax transformation maintains the order of the elements [SB17]:

Policy-based methods follow the objective to maximize the expected return G_t by identifying an optimal parameterization of a policy. Accordingly, the cost function $J(\theta)$ introduced in section 2.2.3 will be used to evaluate the parameterization and the policy. For an episodic task, this measure is defined as

$$J(\theta) = v_{\pi_\theta}(S_0) = \mathbb{E}_\theta\Big[G_t\Big], \tag{2.38}$$

where $v_{\pi_\theta}(S_0)$ is the value of the initial state of an episode. This group of algorithms makes use of the learning signal provided by gradients. The parameterization of the neural network will be updated by

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta_t} \hat{J}(\theta_t), \tag{2.39}$$

where $\nabla_\theta \hat{J}(\theta)$ is the gradient of the cost function and α is a learning rate [SB17]. The challenge of gradient-based policy search is how to estimate the gradient of the performance measure with respect to the parameterization. Fortunately, the policy gradient theorem solves this challenge [Sut+99]. Based on the assumption that the gradient of the cost function equals the gradient of the true value function of the parameterized policy, it can be shown that:

$$\begin{aligned} \nabla J(\theta) &= \nabla v_\pi(S_0) \\ &= \mathbb{E}_\pi \left[G_t \nabla_\theta \log \pi(A_t|S_t,\theta) \right]. \end{aligned} \tag{2.40}$$

Appendix A.1 provides an excerpt of the proof of the policy gradient theorem. The gradient of the cost function is now described as the expectation of the return at time step t times the gradient of the log probability that an action is selected based on a state s_t and the parameterization θ. The *REINFORCE* algorithm [Wil92] makes use of this identity to update its parameterization by

$$\theta_{t+1} = \theta_t + \alpha\gamma^t G_t \nabla_\theta \log \pi(A_t|S_t,\theta), \tag{2.41}$$

where α is a learning rate, γ is a discount factor and $\pi(A_t|S_t,\theta)$ is a stochastic policy. Algorithm 1 shows the usage of this update rule in the pseudo-code of the *REINFORCE* algorithm.

Algorithm 1: *REINFORCE* algorithm [SB17]

```
1 function REINFORCE ($\pi(a|s,\theta)$) :
      Input: a differentiable policy parameterization $\pi(a|s,\theta), \forall A, S_t \in \mathcal{S}, \theta \in \mathbb{R}^{d'}$
      Local Variables: $G$ as the return between $t$ and the end of the episode.
2     Initialize policy parameter $\theta$
3     repeat
4         Generate an episode $S_0, A_0, R_1, ...S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\theta)$
5         foreach step $t = 0, ..., T-1$ of the episode do
6             $G \leftarrow \sum_{t'=t}^{T} R_{t'}$
7             $\theta \leftarrow \theta + \alpha\gamma^t G \nabla_\theta \log \pi(A_t|S_t,\theta)$
```

A likely extension of the *REINFORCE* algorithm in terms of generalization is to substract a baseline $b(S_t)$ from the return G_t. Baselines reduce the variance of the gradient and are unbiased. One can use an arbitrary baseline, as long as they do not vary with a [SB17]. The adjusted update rule is given by

$$\theta_{t+1} = \theta_t + \alpha\gamma^t (G_t - b(S_t)) \nabla_\theta \log \pi(A_t|S_t,\theta). \tag{2.42}$$

2.3.3.3 Actor-Critic Methods

One of the main challenges of policy-based methods is to evaluate the goodness of a policy. Actor-Critic methods represent a solution to this problem. They are designed as an interactive process between an actor and a critic (Figure 15).

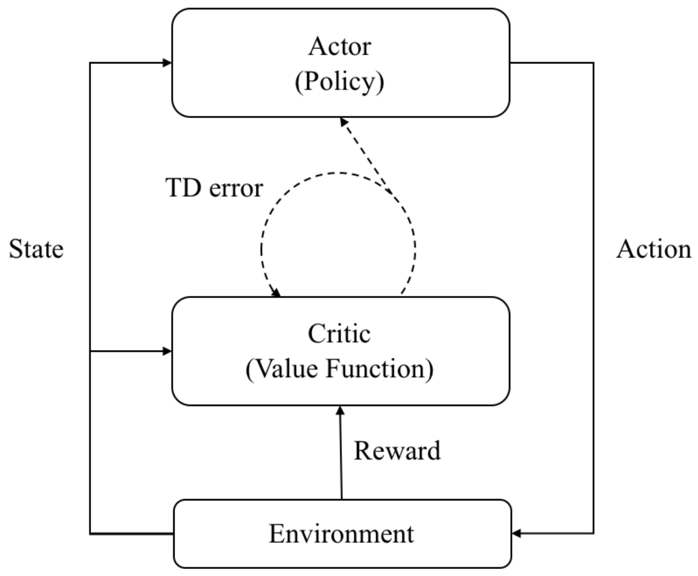


Figure 15: Actor-Critic Set-Up [Aru$^+$17]

The actor is represented by a policy-estimator and will choose an action according to the current state s_t and its internal policy π_θ. After an action A_t was executed by the agent, the environment provides a new state S_{t+1} and a reward signal R_{t+1} to the critic. The critic is defined by an approximated value function $\hat{q}_w$, which is used to evaluate the actors action: $\hat{q}(S_t, A_t, w)$. As soon as this q-value is given to the actor, it will update its parameterized policy. In a next step, the actor produces an action A_{t+1}. Finally, the critic updates its own value function based on $\hat{q}(S_t, A_t, w)$ and $\hat{q}(S_{t+1}, A_{t+1}, w)$. This procedure is repeated in every iteration. Therefore, actor-critic methods feature online learning. Compared to policy-gradient methods with baselines this is a big achievement [Aru$^+$17; SB17].

2.3.4 Challenges

The main challenges for RL agents arise from their limited knowledge about the environment in combination with the objective to maximize cumulated rewards over time. The only way to gather information is to interact with the environment by choosing actions.

2.3.4.1 Exploration-Exploitation Dilemma

In order to gather information, agents need to perform non-optimal actions to explore their environment. This behavior assumes that renouncing on short-term rewards might lead to increased rewards in the long run. In contrast, one can describe exploitation as a greedy behavior which maximizes the rewards based on current information. The resulting conflict is known as the exploration-exploitation dilemma and balancing the trade-off between both is a fundamental challenge for RL agents [KLM96; SB17].

There are different ways to foster exploration. A naive approach which is typically applied in Q-learning is ϵ-greedy. Most of the time, the algorithm exploits its environment by selecting the action which corresponds to the highest Q-value. However, with a probability of ϵ, the model performs a random action and explores the environment [SB17]. In contrast, count-based exploration methods implement an approach that provides incentives to explore regions of high uncertainty [Aru$^+$17; LR85].

2.3.4.2 Credit Assignment Problem

Especially in episodic tasks, RL agents face the problem that rewards materialize delayed after many interactions with the environment. Hence, learning is impeded since the effect of specific actions is blurred. This problem is known as the credit assignment problem [KLM96]. One can consider an episodic game, like e.g. chess. A RL agent would receive a positive reward signal if it wins the game by checkmating the other agents king. If the game results in a draw or loss, the agent would receive a neutral or negative reward. Typically, it takes many moves to end a chess game if the players are on a comparable level. Accordingly, the rewards that RL agents can observe are very sparse and it is unclear which move was the crucial one that decided between win or loss. It is even more likely that the interdependence of multiple moves caused the outcome of the game.

2.3.4.3 Sample Inefficiency

A problem that results from the credit assignment problem and sparse rewards is the sample inefficiency of RL problems. In order to gain actual insights and learn patterns from their environment, agents have to interact with their environment many times. Irpan [Irp18] highlights this problem on the example of the application of DQN architectures in Atari games. State-of-the-art architectures require more than 83 hours of play experience, which equals about 18 million observed frames, to achieve human-like performance. However, humans master this game in a few minutes [Irp18].

Sample inefficiency is a barrier that limits the application of deep reinforcement learning to use-cases, where computation time is no crucial parameter and the cost to generate experience is low. Games represent one of the few use-cases that satisfy this constraint. Due to techniques like self-play models are able to create experience based on a set of rules and simple instructions. However, self-play is no novelty, already Tesauro [Tes92] used it to train TD-Gammon. Recent work of Silver et al. [Sil$^+$16] applied self-play to train AlphaGo Zero.

3 State-of-the-Art

This chapter will provide the state-of-the-art in the domain of end-to-end dialog systems. As mentioned in Section 2.1.3, the third generation of dialog systems is centered around advanced neural networks architectures. Such state-of-the-art models typically build upon a four step framework: **embed, encode, attend** and **predict** [Hon16].

First, Seq2Seq models will be introduced as the core component of the framework. Later on, the section will explain the additional concepts, word embeddings and attention mechanisms, which are required to perform step one and three of the framework. Both concepts represent extensions to the standard Seq2Seq model. While this chapter focuses on the introduction of the underlying concepts of the framework, Chapter 4 will discuss how the different components of the framework work together in this thesis.

In addition to the remarks on the embed, encode, attend and predict framework, this section will provide an overview about the interaction between dialog systems and external knowledge bases. Finally, the selection of an appropriate dataset will be discussed.

3.1 Sequence-to-Sequence Models

Seq2Seq models are an architecture which solves a limitation of ANNs while working with sequential data. Due to their design, neural networks require inputs and outputs to have a fixed dimensionality. However, sequence tasks like machine translation or speech recognition are problems where sequence length is not known a-priori. Seq2Seq models employ RNNs to overcome this limitation [SVL14].

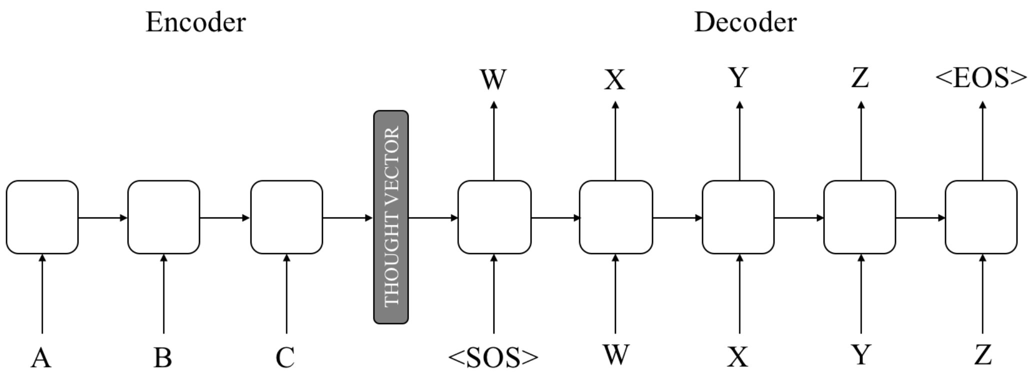


Figure 16: Seq2Seq Model [Cho+14]

Cho et al. [Cho+14] proposed an encoder-decoder design that consists of two RNNs (Figure 16) and applied it to translate from English to French. The first RNN processes an input sequence into a continuous representation, also called **thought vector**. While the input sequence can be of variable length, the size of the thought vector is fixed. One can think of this representation as a summary of the input, which preserves semantic and syntactic information. As soon as the first RNN terminates to read the input sequence, the thought vector is passed to the second RNN. This starts working based on a special start-of-sequence symbol (SOS). Given the fixed-size thought vector, the decoder RNN will generate an output sequence of variable length. The outputs predicted at time step t are fed back into the model at $t + 1$. The decoder terminates as soon as it predicts the end-of-sequence symbol (EOS) [Cho+14].

Sutskever et al. [SVL14] combine the encoder-decoder approach with the usage of LSTM-cells in a deep architecture. Like Cho et al. [Cho+14], they chose a machine translation task. Their model was the first pure neural translation system to outperform a statistical machine translation baseline significantly. However, this success comes at high computational costs. An implementation of their Seq2Seq model with four hidden layers, 1000 cells in each layer, was trained on 12 million sentences. It took eight GPUs ten days of training time. This equals a translation rate of 6,300 words per second [SVL14]. This example highlights that neural architectures are indeed powerful models, but the success of their employment is accompanied by the availability of large amounts of data in addition to massive computational ressources.

3.2 Word Embeddings

The application of neural methods in natural language tasks requires a transformation of words into a numerical representation. Traditionally, this transformation yields a discrete representation, where every word refers to a unique index in a large vocabulary. All but one of the vector values are zero, a characteristic, which accounts for the name **one-hot encoding**. A result of this representation is that the distance between two words in vector space is always the same. Therefore, one-hot encodings lack to represent information about similarities between different words, as their encoding is arbitrary and relations between words is typically evaluated on their distance or the angle between them. This leads to the fact that such discrete word vectors do not generalize across words [Ben+03; PSM14].

Continuous word vectors escape this limitation, because using a real-valued vector instead of a one-hot encoding resolves the issue of equal distances and reduces the dimensionality of the representation significantly. According to Jurafsky and Martin [JM17], word embeddings tend to have a dimensionality between 50 and 500. Sutskever et al. [SVL14] used a 1000-dimensional word embedding in their machine translation task. However, using a 1000-dimension real-valued vector instead of an one-hot vector, with a dimensionality of the vocabulary size $\mathcal{V}$, is an achievement regarding computational efforts and the curse-of-dimensionality, which describes the phenomenon that data becomes sparse as the dimensionality increases [GBC16]. Furthermore, continuous representations can capture semantic and syntactic information [CW08; Mik$^+$13a].

One can distinguish between two families of state-of-the-art approaches that train word embeddings. While GloVe (global vectors) [PSM14] is an example of the methods building on global co-occurrence of words in a corpus, word2vec [Mik$^+$13b] is a representative of the methods that rely on local context windows. Both families offer a way to incorporate pre-trained word embeddings into a neural dialog system. Subsequently, this section will discuss GloVe, as it will be applied in this thesis.

GloVe is an unsupervised learning algorithm, which explicitly aims to map similar words to similar regions in vector space. For this purpose, the algorithm iterates through a corpus and constructs a co-occurence matrix for a predefined vocabulary. The co-occurence matrix X_{ij} captures how often the word i appears in the context of word j. In their experiments, the authors evaluated different corpora, varying from Wikipedia excerpts of 1 billion tokens to web data from Common Crawl with 42 billion tokens. Except for the Common Crawl data the vocabulary was build of the 400,000 most frequent words [PSM14]. They propose a weighted least squares regression, which minimizes

$$J = \sum_{i,j=1}^{\mathcal{V}} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2, \tag{3.1}$$

where w_i and b_i represent the embedding and bias for word i, $\tilde{w}_j$ and $\tilde{b}_j$ are the context word vector and bias of word j. Furthermore, f is defined as a weighting function, which prevents from learning uncommon word pairs. By this design, the authors intend to learn word vectors based on ratios of co-occurrence probabilities. The following example will explain this underlying assumption. Given a probe word k, one would like to assess whether it maps close or far from a target word t. Therefore, Table 2 provides the co-occurrence probabilities $P(k|t)$ of different probe words k given a target word t [PSM14].

Probability and Ratio	k=*solid*	k=*gas*	k=*water*	k=*fashion*
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Table 2: Co-occurrence Probabilities [PSM14]

As *solid* describes the thermodynamic phase of *ice* it seems reasonable that their word vectors should map closer together than the ones of *ice* and *gas*. This intuitive idea is confirmed by the co-occurrence probability of those words. As can be seen, the word *ice* appears more often in relation with *solid*, than it does with *gas*. Furthermore, *gas* co-occurs more frequently with *steam* than it does with *ice*. The ratio of both co-occurrence probabilities given a probe word k captures the addressed relation between the probe word and the two target words. For instance, a large ratio (much greater than one) indicates that a probe word correlates with ice, whereas a small ratio (much smaller than one) indicates that a probe word correlates with $steam$. Furthermore, a ratio of probabilities close to one can be used to identify non-discriminative words, like $water$ and $fashion$. While $water$ co-occurs frequently with both target words, $fashion$ is not related to them and therefore co-occurs infrequently. The authors leveraged this capability of the ratio of probabilities and derived the objective function presented in Equation 3.1 [PSM14].

As a result, Pennington, Socher and Manning [PSM14] found that GloVe outperforms other word embeddings in finding word analogies, named entity recognition and word similarity tasks. GloVe vectors capture several levels of semantic and syntactic information of language. Figure 17 shows a 2-dimensional visualization, where GloVe embeddings preserve complex relations like gender specific terms [PSM14].

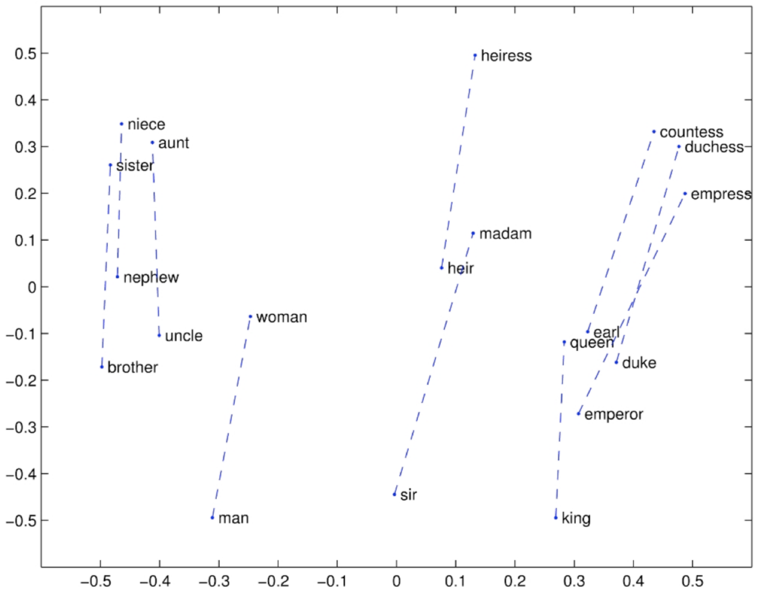


Figure 17: A 2-dimensional GloVe Visualization [PSM14]

GloVe captures the relation between male and female nouns. In each pair, the male phrase is located below the female phrase. The connection between both words points in the same direction and seems to have a similar length. Furthermore, one can observe that pairs group by context. The value-pairs located in the interval [-0.5, -0.3] on the x-axis seem to belong to the category family members. Whereas the group located in the interval [0.3, 0.5] on the x-axis associate nobleman and -woman.

3.3 Attention

Seq2Seq models resolve the problem that the length of input and output sequences needs to be defined a-priori. However, the encoder-decoder approach created a new bottleneck, the fixed size thought vector. While a network is processing longer sequences, it might be difficult to encode all relevant information into such representation [BCB14].

Bahdanau, Cho and Bengio [BCB14] addressed this problem in a machine translation task and proposed an attention mechanism to improve performance. Attention in neural networks originated in the field of image recognition, where it was used to let a network focus on certain regions of an image [Xu+15]. In natural language processing however, the attention mechanism leverages encoder hidden states to decide to which inputs it should focus during decoding.

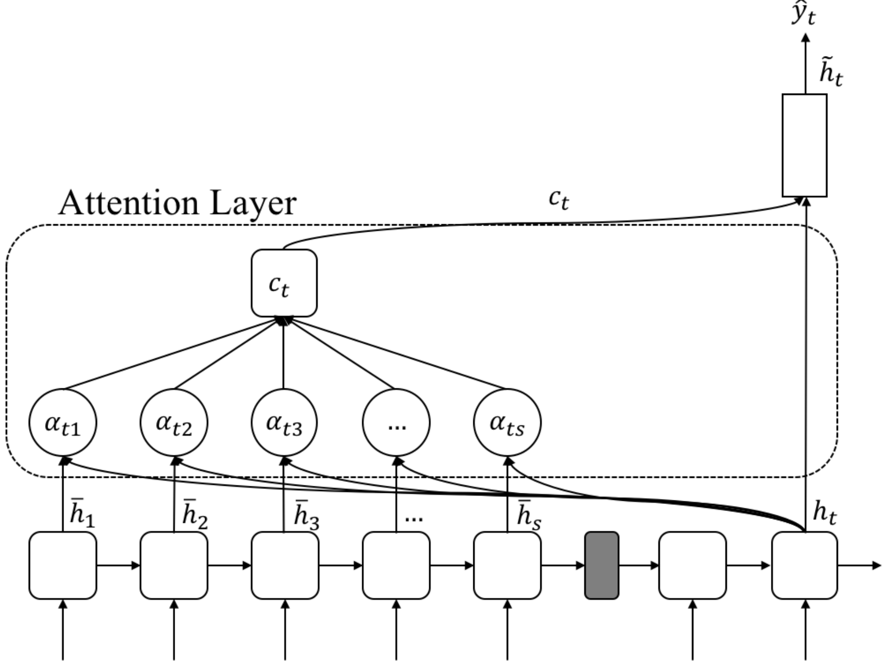


Figure 18: Seq2Seq Model augmented with an Attention Mechanism [LPM15]

Figure 18 illustrates the behavior of the attention mechanism to predict the output $\hat{y}_t$. Like in the standard Seq2Seq model, the first RNN produces a fixed size representation based on the input sequence. However, at each encoder time step s, the models stores the RNNs hidden state $\bar{h}_s$. After the input sequence terminated, the encoder passes the thought vector to the decoder. During decoding, the second RNN scores the current decoder (target) hidden state h_t against all encoder (source) hidden states $\bar{h}_s$. In their initial proposal, Bahdanau, Cho and Bengio [BCB14] defined the score function as

$$score(h_t, \bar{h}_s) = v_a^T \tanh(W_a[h_t, \bar{h}_s]), \tag{3.2}$$

where v_a and W_a are weights and the score function concatenates the source hidden state $\bar{h}_s$ and target hidden state h_t. Therefore, this will be referred to as the **concat** scoring function. Later work by Luong, Pham and Manning [LPM15] refined it based on the dot product and defined it as

$$score(h_t, \bar{h}_s) = \begin{cases} h_t^T \bar{h}_s & \text{dot} \\ h_t^T W_a \bar{h}_s & \text{general} \end{cases}. \tag{3.3}$$

This thesis will use the initial scoring function by Bahdanau, Cho and Bengio [BCB14]. However, some findings about the refined scoring function will be discussed at the end of this section. In general, the scoring function is implemented with the goal, to yield high values if a source hidden state $\bar{h}_s$ is important to the decoding of the target hidden state h_t. Based on the scoring function one can compute the attention weights α_{ts} by

$$\alpha_{ts} = \frac{\exp(score(h_t, \bar{h}_s))}{\Sigma_{s'=1}^{S} \exp(score(h_t, \bar{h}_{s'}))}. \tag{3.4}$$

Due to the application of the softmax transformation, the attention weights $\alpha_{t1}, ..., \alpha_{ts}$ between a certain target hidden state h_t and all source hidden states $\bar{h}_1, ..., \bar{h}_s$ sum to one and can be seen as a probability distribution. This distribution is then used to determine a context vector c_t, which is defined as a weighted sum:

$$c_t = \sum_s \alpha_{ts} \bar{h}_s. \tag{3.5}$$

The context vector can be described as a summary of relevant inputs with regards to a specific decoder time step t. In order to compute the prediction $\hat{y}_t$ it is necessary to combine the information of the source-side context vector c_t with the current decoder hidden state h_t. This combination is defined as the attentional hidden state

$$\tilde{h}_t = \tanh(W_c[c_t, h_t]), \tag{3.6}$$

where W_c is a weight matrix applied on the alignment of the context vector c_t and the hidden state h_t. The attentional hidden state $\tilde{h}_t$ is then used to compute

$$\hat{y} = \text{softmax}(W_y \tilde{h}_t), \tag{3.7}$$

where W_y is a weight matrix and a softmax transformation turns the output into a probability distribution [BCB14; LPM15].

Bahdanau, Cho and Bengio [BCB14] found that the application of an attention mechanism leads to improved performance compared to conventional Seq2Seq models. Especially the performance in longer sequences improved and made the model more robust. Thereby, attention resolves the bottleneck between encoder and decoder. However, attention is computationally expensive, because at each decoder time step t the algorithm computes the attention over all source hidden states $\bar{h}_s$. This is impractical while working with longer sequences like paragraphs or full documents. Therefore, Luong, Pham and Manning [LPM15] introduced the concept of local attention. Instead of taking all encoder hidden states into account, they propose to focus on a small window of source hidden states. Indeed, they observed that local attention was able to decrease the test costs. Furthermore, they showed that the choice of the score function has an impact on the performance. The dot score function worked best for global attention, whereas the general score function performed better for the local attention [LPM15].

Another representative of attention mechanisms are pointer networks as proposed by Vinyals, Fortunato and Jaitly [VFJ15]. Pointer attention is able to solve problems where the output dictionary corresponds to elements of the input sequence. The approach eases the solution of combinatorial problems with variable length of the output sequence. They leverage attention weights α_t to define the probability of the next output token as

$$p(C_t|C_1, ..., C_{t-1}, x) = \alpha_t, \tag{3.8}$$

where x is an input sequence with n elements and C_i represents the i-th element in the output sequence, which is an index between 1 and n. A major advantage of pointer attention is that it reduces the size of the output space significantly [VFJ15].

3.4 Knowledge Base Interaction

Chapter 2.1 introduced dialog systems as models, which leverage dialog history to produce a natural language response during the interaction with another agent. Nevertheless, a dialog system can use more resources than current and previous user utterances. Many use-cases require the utilization of external knowledge [Ser+15]. For example, a dialog system, which assists during the booking process of an accommodation, needs access to available hotels and hostels in the area of interest. Likewise, it is possible for a conversational agent to use news articles or user reviews to improve the interaction with the other agent. However, this work will focus on the incorporation of knowledge that is stored in a database. Subsequently, this section distinguishes between two classes of dialog systems, based on how they interact with a KB. Afterwards, the term NoSQL will be discussed in order to introduce Elasticsearch. Elasticsearch is a search engine that will be used as a NoSQL database, storing the additional meta-information to the dataset.

3.4.1 Hard-KB Lookup

One can distinguish between two classes of dialog systems based on the way how they interact with a knowledge base [EM17]. According to Dhingra et al. [Dhi^{+}16] dialog systems that create a semantic representation out of a natural language question and operate it against a database perform **hard-KB lookup**.

Recent work employed sketch-based approaches to produce SQL queries out of natural language questions [XLS17; Yag17]. Both approaches require human intervention to scale up to other query languages. This is due to the fact that they make strong assumptions about the structure of SQL and hand-engineering is necessary to adjust the sketch. In contrast, there is research aiming for more generic models in order to foster generalization across different semantic representations. For instance, Dong and Lapata [DL16] implemented an attention augmented Seq2Seq model and achieved competitive results through a variety of semantic representations. Bordes, Boureau and Weston [BBW17] applied memory networks (MemNNs) to generate API calls out of dialogs related to restaurant bookings. MemNNs incorporate the capability to read and write to a memory module [Suk^{+}15; WCB14].

The previously discussed models share the same shortcoming, they require supervision before they actually query the database. This is due to the fact that operating queries against the database is non-differentiable. This is a major drawback of hard-KB lookup as it impedes the application of the models in an end-to-end fashion. Wen et al. [Wen^{+}16] proposed a modular end-to-end dialog system. As they use hard-KB lookup, they needed to train the different components of their model separately. In order to train the components, the model requires intermediate labels, which leads to increased costs. Another major drawback of hard-KB lookups is that the results returned from the database no longer represent the uncertainty in the underlying parsing problem [Che^{+}17].

A way to overcome the problem of non-differentiability is to use reinforcement learning to include feedback from the database response [Li^{+}17; WZ16; ZXS17]. Li et al. [Li^{+}17] proposed an end-to-end neural dialog system, which is based on the standard pipeline architecture. Their architecture includes a dialog management component. Such combines the components of the dialog state tracker and the dialog response selector of the architecture discussed in Section 2.1.2. The dialog manager was implemented as a Deep Q-Network, which is a value-based method according to the classification in Chapter 2.3.3. This design choice distinguishes their model from the approach used in this thesis. Instead of using a value-based method, this work uses a policy-based method inspired by Zhong, Xiong and Socher [ZXS17]. They proposed a model consisting of three individual components to parse natural language questions into SQL queries. While they used supervised learning to predict an aggregator and the select column, the where clause was determined by the application of policy gradient. Instead of using reinforcement learning to achieve an end-to-end character, the authors aimed to improve the quality of the where clause. They state that arguments in the where condition can be swapped. For instance, "WHERE A=a AND B=b" would yield the same result like: "WHERE B=b AND A=a". While supervised learning would evaluate the query based on the string match, reinforcement learning is able to address this unordered nature of the where clause [ZXS17].

3.4.2 Soft-KB Lookup

Despite hard-kb lookup, there are dialog systems that perform **soft-KB lookup**. Such systems compute a probability distribution over the knowledge base and therefore work in an end-to-end setting [EM17]. Dhingra et al. [Dhi+16] proposed a neural end-to-end approach which interacted with real users via Amazon Mechanical Turk. Based on a tabular representation $\mathcal{T}$ of the knowledge base they determine the probability that a user is interested in a certain entry $\mathcal{T}_{ij}$, where i corresponds to a specific entity and j refers to a category. Their implementation uses a GRU as policy network, which is trained by policy gradient. The authors observed that their model tended to fail from random initialization. They attribute this to the credit assignment problem. In order to overcome this problem they use imitation learning to mimic hand-crafted agents [Dhi+16].

Dodge et al. [Dod+15] proposed the application of MemNNs in soft-KB lookup. They used the memory component of the network to store each of the entries of the knowledge base individually and train the model to attend over the memory entries. Furthermore, Bordes et al. [Bor+15] showed that their MemNN model was able to generalize on unseen data. They observed that MemNNs were able to handle additional facts from another knowledge base, which were added after training. Despite the fact that the model was not re-trained, the MemNN achieved remarkable results which were close to the state-of-the-art [Bor+15]. Eric and Manning [EM17] developed an LSTM encoder-decoder model and applied it in a in-car personal assistant task. Based on the concept of key-value memory networks [Mil+16], they implemented a key-value approach to score the target hidden state against the keys of the KB entries during decoding. Since the KB is stored in triplets, the key of each entry is computed as the sum of the word embeddings of the subject and the relation. Due to this mechanism they were able to retrieve relevant entries from the knowledge base. The authors found that their model outperforms competitive heuristics and neural baselines, like a Seq2Seq model augmented with attention [EM17].

However, soft-KB lookups have one drawback, as such dialog systems compute a distribution over every entry in the KB, they can cause high computational costs. These costs depend on the underlying mechanism to compute the distribution, as well as the size and design of the KB.

3.5 NoSQL and Elasticsearch

The utilization of information is a key capability of today's business world. In 1970 Codd [Cod70] proposed the idea of relational databases. Furthermore, he was involved in the development of SQL, which became the predominant technology to store and access structured data in relational databases. However, recent developments show an increased interest in NoSQL databases. NoSQL represents an abbreviation for "not only SQL databases". This name suggests that NoSQL databases do not aim to replace relational database structures like SQL, but to complete the set of database tools for tasks, where SQL is not the optimal choice [NPP13].

While NoSQL databases are able to handle e-mails, social media contents or images, relational databases have problems with such unstructured data types. This is caused by several reasons. First, relational databases rely on predefined table structures in that unstructured data do not fit naturally. Fitting unstructured data into a predefined frame binds resources and results in complex database structures, which makes the database slow to work with. Non-relational databases do not have this problem, they can handle unstructured data directly [Lea10]. Second, the performance of relational models decreases as the data volume increases. Relational databases are not designed to work on distributed systems. Therefore, their scalability is limited to an increase of the computational power on a single system. However, many NoSQL architectures are designed to guarantee high availability and high scalability due to their distribution across multiple server. Thereby, NoSQL architectures offer a significant advantage over relational databases. [NPP13].

By the time of writing, the TOP 10 of the database-engines popularity ranking consisted of six relational databases and four non-relational databases. The ranking is based on a set of parameters, like Google trends or the number of times a systems name was mentioned in social media, to evaluate the general interest in the system. Furthermore, job offers and profiles in professional networks that include the name of a database, as well as the amount of related topics on technical discussion websites, are used to capture the technical importance of a system [Sol18]. The ranking shown in Table 3 reflects what was mentioned before. In general, relational databases are a powerful tool if the available data is structured. This accounts for the many use-cases in the business world. However, there are situations, where data is unstructured and relational databases do not work well. In such situations non-relational databases offer a useful alternative [Lea10].

Elasticsearch is one of the non-relational approaches that made it into the TOP 10 of the ranking. Elasticsearch belongs to the category of search engines, which can be seen as a subgroup of non-relational databases. Like other non-relational databases Elasticsearch brings the advantage of high availability and high scalability. Furthermore, it features

Rank	Name	Database Type
1.	Oracle	relational
2.	MySQL	relational
3.	Microsoft SQL Server	relational
4.	PostgreSQL	relational
5.	MongoDB	non-relational
6.	DB2	relational
7.	Microsoft Access	relational
8.	Elasticsearch	non-relational
9.	Redis	non-relational
10.	Cassandra	non-relational

Table 3: DB-Engines Popularity Ranking - April 2018 [Sol18]

document-oriented storage and full-text searches in near real-time. This makes Elasticsearch to a state-of-the-art system which rapidly gained importance over the last years [Ama18].

This thesis will use Elasticsearch to store the knowledge base which comes with the dataset. Despite its technical relevance, there are additional reasons for this decision. First, there are several approaches which aim to parse natural language questions into SQL queries [XLS17; Yag17; ZXS17]. This work aims to extend this line of research to NoSQL databases. Second, inovex GmbH belongs to one of the first partners of Elasticsearch in Germany. For this reason, this work can build upon expertise and know-how. Finally, the Elasticsearch framework offers standard interfaces which eases implementation. However, this work will not discuss technical features of Elasticsearch.

3.6 Datasets

This section will discuss the selection of the dataset used in this thesis. In order to develop an end-to-end goal-driven dialog system this thesis will focus on written dialog tasks. This scope intends to limit the complexity of the problem setting by excluding additional tasks like e.g. speech recognition.

Serban et al. [Ser+15] provide a survey of available corpora to develop data-driven dialog systems. They focus on publicly available datasets. In addition, the open-source platform ParlAI offers an environment to implement and test dialog systems on several datasets [Mil+17]. Those two resources represent the starting point for the dataset selection process in this thesis.

Miller et al. [Mil+17] group the available datasets on ParlAI in five separate categories: sentence completion, question answering, goal-oriented dialog, chit-chat and visual question answering. This distinction appeared to be a good guidance to conduct a first pre-selection. As this thesis targets for goal-driven dialog, the categories sentence completion and chit-chat were excluded from further consideration. Chit-chat describes conversations that do not follow specific goals and therefore better fit to train non-goal-driven dialog systems. Sentence completion however, is a supportive skill, but does not contribute to the development of an end-to-end dialog system. As visual question answering incorporates the utilization of images it was excluded as well. As a result of this pre-selection, two out of five categories remained relevant: question answering and goal-oriented dialog. Question answering (QA) is a simple dialog task which only involves one interaction turn. It aims to the retrieval of factoid answers based on a specific question. Goal-oriented dialog summarizes different types of conversational elements which are required to fulfill a complex task, for instance booking a flight. In this way, question answering can be seen as a subtask of goal-oriented dialog.

Henceforth, this thesis will focus on question answering as it is one of the building blocks to complete the challenge of full goal-oriented dialog. Table 4 provides details about the size and the incorporation of external knowledge for an excerpt of available question answering datasets in ParlAI [Mil+17].

Dataset	Size (Train / Val / Test)	External Knowledge
bAbI 1k (10k) [Wes+15]	1k (10k) / 0 / 1k (10k)	-
MovieDD-QA [Dod+15]	**96k / 10k / 10k**	**KB**
Simple Questions [Bor+15]	76k / 11k / 21k	KB
SQuAD [Raj+16]	108k	Full text
Web Questions [Ber+13]	3k / 778 / 2032	KB
Wiki Movies [Mil+16]	96k / 10k / 10k	KB and full text
Wiki QA [YYM15]	2k / 296 / 633	Full text

Table 4: Excerpt of available Question Answering Datasets in ParlAI [Mil+17]

Since the bAbI datasets do not incorporate an external KB, they were removed from further consideration. In addition, Wiki QA and SQuAD were excluded based on the type of their external knowledge. Both datasets leverage full text. For the same reason, Wiki Movies was removed since it extends the QA task of the Movie Dialog Dataset (MovieDD) with external knowledge in form of Wikipedia articles [Mil+16]. Out of the remaining datasets, the MovieDD-QA was selected since it combines a domain specific use case with a large scale dataset. While Simple Questions is an open domain dataset, Web Questions was considered to small. Hence, this thesis will conduct experiments on the MovieDD-QA. The dataset will be introduced an analyzed in Chapter 4.2.

3.7 Conclusion of the State-of-the-Art

The previous chapter provided an answer to the first research question by highlighting the state-of-the-art in dialog tasks. In general, dialog can be seen as a task based on sequential inputs, which accounts for the usage of RNNs. Typically, two RNNs are combined to implement an encoder-decoder architecture.

The application of word embeddings enables such methods to transform words into a numerical representation and use them as inputs for computations. Since pre-trained word embeddings capture structural information of language, like semantics or syntax, they contribute to an increased performance. Furthermore, state-of-the-art models make use of attention, which allows them to focus on subsets of the inputs while predicting the outputs. This improves performance especially for longer sequences [BCB14]. Pointer attention mechanisms are used when outputs correspond to positions in the input sequence. Their design significantly decreases the output space and therefore reduces the computational effort [VFJ15].

Additionally, this section discussed how dialog systems interact with external KBs. One can distinguish between hard-KB lookup and soft-KB lookup. While the former produces database queries, the latter computes a probability distribution over the KB entries. Both lookup types have been implemented recently. However, it appears that there is no analysis of their competitiveness on the same dataset. A major shortcoming of hard-KB lookup is the non-differentiability of the database operation. The subsequent section will propose two different ways to train a Seq2Seq model and preserve the end-to-end character of the system.

4 Approach

This thesis develops an end-to-end dialog system that answers natural questions in the movie domain. Therefore, this section outlines the underlying methodology. In a first step, the general approach, an LSTM-based Seq2Seq model, will be introduced. Afterwards, the MovieDD will be analyzed. Finally, the Seq2Seq model will be implemented into an architecture that is adjusted on the MovieDD use case and the interaction with an Elasticsearch instance.

4.1 General Approach

Generally speaking, this thesis proposes a Seq2Seq model, which maps an input sequence x into an output sequence $\hat{y}$. The input sequence

$$x = \Big[x_1, ..., x_S\Big] = \Big[x_1^v, ..., x_m^v, x_{m+1}^c, ..., x_{m+n}^c, x_{m+n+1}^q, ..., x_{m+n+p}^q\Big] \tag{4.1}$$

consists of three subsets. The first m words describe the vocabulary of command tokens. The following n words represent the different categories that exist in the database. Finally, there is a sequence of p words which equals the question that is asked to the dialog system.

The output $\hat{y} = [\hat{y}_1, ..., \hat{y}_T]$ is a sequence of words chosen from the input vocabulary. If the command token <*EOS*> is contained in x^v, the length of the output sequence T is variable. As soon as the model predicts this command token, the decoder terminates. If however, <*EOS*> is not fed into the model as part of x^v, the length of the output sequence T needs to be defined in advance. For each training example x the dataset provides a ground truth y.

Let $query(\cdot)$ denote a database operation, which takes the predicted sequence $\hat{y}$ as input and returns a database response $\omega_i = [\omega_1, \ldots, \omega_\Omega]$. Typically $query(\cdot)$ is non-differentiable. Depending on the choice of the database and the definition of $query(\cdot)$, the database response ω can either be a ranked or an unordered list. A single response element ω_i with $i \in [1, \ldots, \Omega]$ can consist of one or multiple entities e. Furthermore, ψ denotes the ground truth answer to the question x^q. Like the database response elements, an answer ψ can consist of one or multiple entities e.

4.1.1 Encoder

The model is designed as a two-layer encoder-decoder LSTM. In the encoder, a bi-directional LSTM reads the embedded inputs from x_1 to x_S and vice versa. Thereby it computes the hidden states in the first layer of the encoder by

$$\begin{aligned}\overrightarrow{h}_s^{(1)} &= LSTM\left(emb(x_s), \overrightarrow{h}_{s-1}^{(1)}\right)\\ \overleftarrow{h}_s^{(1)} &= LSTM\left(emb(x_s), \overleftarrow{h}_{s+1}^{(1)}\right),\end{aligned} \tag{4.2}$$

where $\overleftarrow{h}_s^{(1)}$ and $\overrightarrow{h}_s^{(1)}$ are hidden states of the forward and backward pass at time step s and $emb(x_s)$ is the embedding of the s-th element of the input sequence x. Furthermore, the hidden states of the second layer are computed by

$$\begin{aligned}\overrightarrow{h}_s^{(2)} &= LSTM\left(\overrightarrow{h}_s^{(1)}, \overrightarrow{h}_{s-1}^{(2)}\right)\\ \overleftarrow{h}_s^{(2)} &= LSTM\left(\overleftarrow{h}_s^{(1)}, \overleftarrow{h}_{s+1}^{(2)}\right),\end{aligned} \tag{4.3}$$

where the outputs of the first layer of the LSTM are fed as inputs into the second layer of the network. At each encoder time step s one can obtain the combined hidden state of the LSTM by

$$\bar{h}_s^{(l)} = \left[\overrightarrow{h}_s^{(l)}, \overleftarrow{h}_s^{(l)}\right], \tag{4.4}$$

where $l \in 1, 2$ represents the layer of the LSTM.

4.1.2 Decoder

In contrast, the decoder is designed as a unidirectional LSTM with two layers. As a result, one can compute the hidden states of the decoder according to

$$\begin{aligned}h_t^{(1)} &= LSTM\left(y_{t-1}, h_{t-1}^{(1)}\right)\\ h_t^{(2)} &= LSTM\left(h_t^{(1)}, h_{t-1}^{(2)}\right).\end{aligned} \tag{4.5}$$

In the first layer of the decoder, the LSTM takes the prediction of the previous time step y_{t-1} as input. In order to compute the hidden states at time step 1, additional remarks are required. In this case, the decoder receives a concatenation of the final encoder hidden states

$$\begin{aligned}h_0^{(1)} &= \left[\overrightarrow{h}_S^{(1)}, \overleftarrow{h}_1^{(1)}\right]\\ h_0^{(2)} &= \left[\overrightarrow{h}_S^{(2)}, \overleftarrow{h}_1^{(2)}\right]\end{aligned} \tag{4.6}$$

as inputs. Since there is no prediction y_0, the model is fed the index of a start-of-sequence signal, which is part of x^v.

4.1.3 Attention Mechanism

Section 3.3 introduced attention mechanisms. This thesis defines the attention scores as

$$score\Big(h_t, \bar{h}_s\Big) = v^T \tanh\Big(W_s\bar{h}_s + W_t h_t\Big), \tag{4.7}$$

where W_s, W_t and v are learnable weight matrices [BCB14]. Hence, the attention weights can be computed by

$$\alpha_{ts} = \frac{\exp(score(h_t, \bar{h}_s))}{\Sigma_{s'=1}^{S} \exp(score(h_t, \bar{h}_{s'}))}. \tag{4.8}$$

Furthermore, one can concatenate the attention weights between a specific decoder hidden state h_t and all encoder hidden states $\bar{h}_1$ to $\bar{h}_S$ to a single vector:

$$\alpha_t = \Big[\alpha_{t1}, ..., \alpha_{ts}\Big]. \tag{4.9}$$

4.1.4 Training with Intermediate Labels

This section discusses how the proposed Seq2Seq model is trained with intermediate labels. Since the execution of the database queries breaks the differentiability, intermediate labels are used in order to provide feedback about the correctness of the produced queries. However, this way of training the model does not include feedback from the retrieved responses after operating the queries.

At each decoder time step t, the output of the model is computed by

$$\hat{y}_t = \text{argmax}(\alpha_t). \tag{4.10}$$

The predicted output sequence $\hat{y}$ and the corresponding ground truth y are used to train the model with cross entropy loss as presented in Section 2.2.3.1.

4.1.5 Training with Policy Gradient

Leveraging the responses of database operations is possible by training the proposed Seq2Seq model with policy gradient, as introduced in Section 2.3.3.2. In this case, the attention weights are used to formulate a stochastic policy

$$\pi\big(\hat{y}_t \mid \hat{y}_1, ..., \hat{y}_{t-1}, x, \theta\big) = \alpha_t, \tag{4.11}$$

which defines the probability of choosing the next token of the output sequence $\hat{y}_t$ [VFJ15]. Hence, action a_t corresponds to the selection of a specific word of the input vocabulary, whereas state s_t is defined by previous predictions $\hat{y}_0$ to $\hat{y}_{t-1}$, the set of command tokens

x^v, the database categories x^c and the posed question x^q. Furthermore, the network parameterization is given by θ.

The reward function $R = \Big[-2, -1, +z\Big]$ is defined as

$$R = \begin{cases} -2 + R^+, & \text{if } query(\hat{y}) \text{ is an invalid query} \\ -1 + R^+, & \text{if } query(\hat{y}) \text{ is a valid query but yields an incorrect result} \\ +z + R^+, & \text{if } query(\hat{y}) \text{ is a valid query and yields the correct result,} \end{cases} \tag{4.12}$$

where z is a positive reward that will be varied during the experiments. A query is considered a valid query if its operation yields a non-empty response and causes no error. Furthermore, a correct result matches at least one of the ground truth entities of the answer. R^+ is a count-based exploration bonus, which is defined by

$$R^+ = \begin{cases} \sum_{t \in T^{R^+}} \sqrt{\dfrac{2 * \log(n)}{count(\hat{y}_t)}} & \text{if boni are active} \\ 0 & \text{else,} \end{cases} \tag{4.13}$$

where n is the number of samples in a mini-batch and $count(\hat{y}_t)$ is the number of times that action $\hat{y}_t$ was chosen at time step t of the mini-batch iteration. Due to the sequential character of $\hat{y}$, the exploration bonus is accumulated over a set of time steps $t \in T^{R^+}$. The design of the exploration bonus is inspired by upper confidence bounds [LR85]. Since its application is optional, a hyperparameter is defined to decide whether to use the boni or not.

Finally, this variant of the model is trained with the *REINFORCE* algorithm according to the following update rule:

$$\theta_{t+1} = \theta_t + \alpha R \nabla_\theta \sum_{t \in T} \log \pi \Big(\hat{y}_t \mid \hat{y}_1, ..., \hat{y}_{t-1}, x, \theta\Big). \tag{4.14}$$

Due to the episodic character of the setting, the discount factor γ is set to one and omitted in the equation above.

4.2 Movie Dialog Dataset - Question Answering

This section provides an insight into the MovieDD, which was introduced to foster the development of end-to-end dialog systems [Dod+15]. The dataset consists out of five subtasks, each of them addressing different functionalities of such dialog systems. This thesis works with the QA task of the MovieDD, which is build to determine the capabilities of a conversational agent to answer factoid questions. The MovieDD-QA task is divided two complementary resources, a KB and a dataset that consists of natural language question-answer pairs. The following subsection will analyze the knowledge base, which incorporates information about 17,340 movies. Furthermore, it will describe the adjustments that were made to store the knowledge base into an Elasticsearch instance. Afterwards, the dataset will be examined and it will be used to generate intermediate labels in order to train on question-query pairs.

4.2.1 Knowledge Base

The main resource for the development of the KB was the Open Movie Database (OMDb). In total, the dataset contains information about 17,340 movies. Each movie is connected to a set of meta data, which was sourced from OMDb. In addition, the authors augmented the dataset with tags related to the movies. Those were retrieved from MovieLens [Dod+15].

The collected data is stored in triples, like

$$(\underbrace{\text{The dark horse}}_{subject}, \underbrace{\text{starred\_actors}}_{relation}, \underbrace{\text{Bette Davis}}_{object}), \tag{4.15}$$

where the subject refers to a movie title throughout the KB.

4.2.1.1 Adjusting the KB Design

Since this thesis uses an Elasticsearch instance to store the KB, it was necessary to adjust the design of the knowledge base. Therefore, a unique index was created for each movie. Each index stores the information about a movie in different categories. In general, a category corresponds to the relation in the original design of the KB. Despite the relation types, there is one additional category that stores the name of a movie for each index. In total, the KB contains 11 different categories. Table 5 shows index 8,741, which stores the movie "Live Free or Die Hard". As one can see, it contains information about all but two categories.

Category	Content
Name	Live Free or Die Hard
Actors	Bruce Willis
Author	John Carlin
Director	Len Wiseman
Genre	Action
Rating	Good
Votes	-
Language	-
Plot	JohnMcClane and a young hacker join forces to take down master cyber-terrorist Thomas Gabriel in Washington D.C.
Release Year	2007
Tags	Action, Bruce Willis, Computers, Good, United States

Table 5: Exemplary KB Entry (Index 8,741)

4.2.1.2 Descriptive Analysis of the Knowledge Base

As mentioned before, the adjusted KB contains 11 different categories and stores 17,340 movies. Figure 19 shows how frequent the different categories appear in the knowledge base. Almost all indexes contain information about movie name, plot, year and director, whereas the categories language, rating and votes appear very seldom.

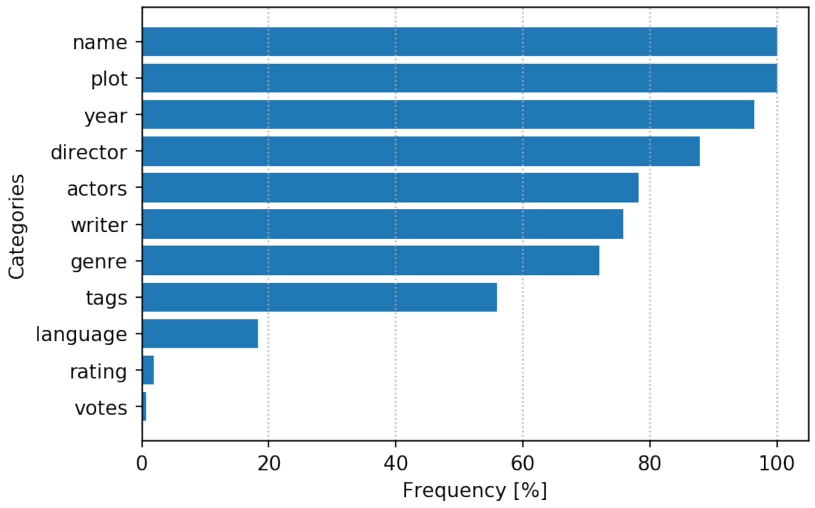


Figure 19: Frequency of Categories throughout the KB

This implies that the quality of information about movies differs. In fact, the average number of categories per index is 6.87 and the standard deviation is 1.17. While 82.0% of the indexes contain between 6 and 8 categories, about 3.1% contain 4 or less categories. Indeed, there are very few indexes (0.02%) with 10 categories and none with 11 categories.

4.2.2 Datasets

The Movie Dialog Dataset QA task is based on the Simple Questions dataset by Bordes et al. [Bor+15]. Dodge et al. [Dod+15] identified a subset of relevant questions from Simple Questions and enlarged it with data from the OMDb. The resulting dataset is divided in a training, validation and test split of approximately 96k, 10k, 10k examples, respectively [Dod+15].

4.2.2.1 Terminology and Introduction of the Running Example

This subsection will present the approach specific terminology and introduce the running example in order to improve the understandability. In Section 4.2.3.1 this terminology will be used to explain the production of intermediate labels. This thesis assumes that one can capture the meaning of each question with three parameters. Such is possible due to the clearly defined use-case and the low complexity of the questions. Hence, applying the model on other datasets would require to verify the validity of this assumption.

The original dataset provides each training example as a question-answer pair. For instance, the question "who was the writer of True Lies?" can be answered with the screenwriters *James Cameron* and *Claude Zidi*. Table 6 shows how the different concepts apply to the example. The first parameter, **query entity** (QE), refers to the object in a question, which is the movie *True Lies*. Since there is no restriction about the length of a movie title or other categories, the number of words captured by the QE can vary. The second parameter, **query field** (QF), captures the category of the QE. As mentioned before, True Lies is a movie *name*. Finally, the **response field** (RF) specifies the requested category of the database response. Both fields are restricted to contain only one category.

	Running Example
Question	Who was the writer of True Lies?
Answer	James Cameron, Claude Zidi
Query Entity	True Lies
Query Field	Name
Response Field	Author
Question Pattern	Who was the writer of [@name]?
Question Class	Name to Author

Table 6: Running Example

A **question pattern** refers to the generic form of a question. Instead of a specific entity, the question pattern contains a category placeholder, e.g. *[@name]*. Thereby, it can be used to summarize questions that only differ in their entity. Furthermore, a **question class** is defined as the set of questions patterns that share their QF and RF.

4.2.2.2 Descriptive Analysis of the Datasets

In order to determine how demanding the MovieDD-QA task is, this section analyzes the variety of questions in the dataset. Therefore, Figure 20 reports the distribution of first words of a question. It was found that *what* and *who* account for more than two-thirds of questions. Both words appear to be flexible with respect to corresponding question patterns and question classes. For instance, *what* can either be found in: "what rating did [@movie] get?" or in: "what movies did [@actors] act in?", whereas *who* can be used to address different RFs like *author*, *director* or *actors*.

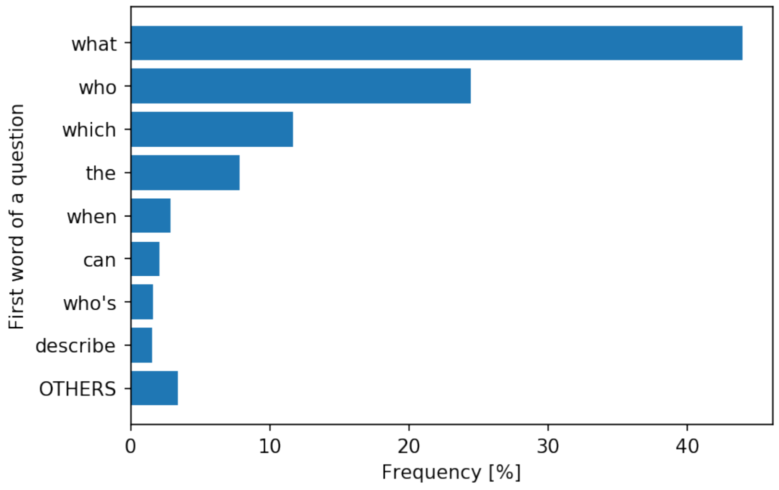
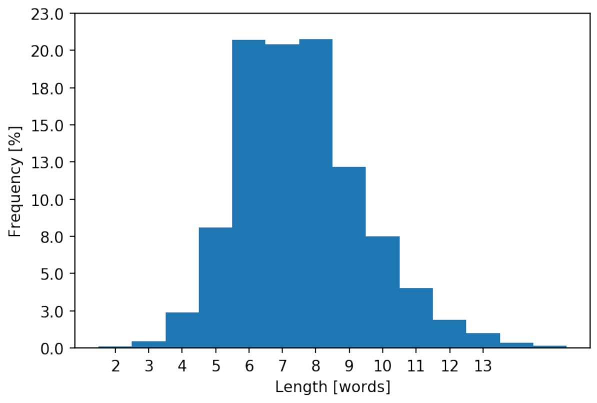


Figure 20: First Words of Questions in the Development Set.

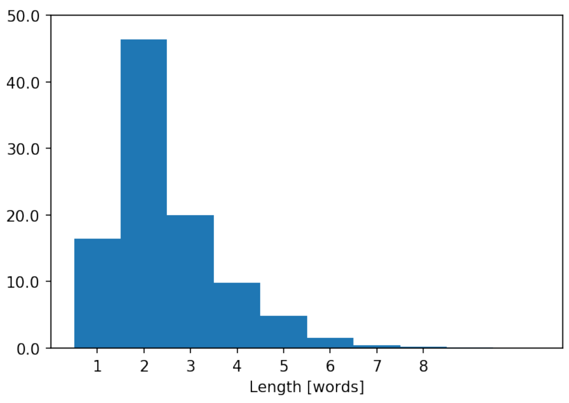
While most of the less frequent words share this characteristic, it was found that *when* is an identifier for questions that aim to retrieve the release *year* of a movie. First words of a question, which occur less than 100 times in the dataset, are combined in *OTHERS* and account for 3.4% of the dataset.

In addition, Figure 21 presents (a) the frequency of the lengths of the questions and (b) the length of the entities that appear in these questions. The average question in the development set is 7.58 words long and the standard deviation is 1.94. About 73.9% of question lengths fall into the interval between six and nine words. However, there are a few outliers. On the one hand, "who directed Cowboy?" is a representative of 0.5% of the questions with less than four words. On the other hand, 1.7% of the questions consist of more than 13 words, like: "what was the release year of Who is Harry Kellerman and why is he saying those terrible things about me?" The average entity length is 2.49 words and the corresponding standard deviation is 1.23. A significant share of entities with the length two is caused by the question classes which QFs correspond to a person, like *author* or *director*.

Figure 20 and 21 show results that were observed on questions of the development set. Since the training and test set show similar patterns they are not reported.



(a) Questions



(b) Entities in Questions

Figure 21: Distribution of Question and Entity Lengths in the Development Set

In contrast, Table 7 shows a more abstract contemplation on the level of question classes. Each row in the table corresponds to a question class, which combines all question patterns that share the same QF and RF. In total, the dataset consists of 13 different question classes and 160 question patterns. Although, *name to rating* and *name to votes* only appear in the training set. As one can see, the number of question patterns per class varies between 7 and 20. There are four classes that consist of less than 10 question patterns. The frequency of the question classes appears to be very balanced across the three splits of the dataset. However, within a single split of the dataset the frequency of question classes is skewed.

Note that every question class contains the category *name* either as its QF or its RF. Future work might adjust the dataset in order to remove this over-representation. One can think of many examples that combine different categories besides the movie title. For instance, "which actors worked with [@director]?" or "which author is known to write [@genre] movies?".

Query	**Response**	**#Question**	**Frequency**		
Field	**Field**	**Patterns**	**Train**	**Dev**	**Test**
Name	Year	14	14.40%	14.63%	14.27%
Name	Director	14	13.10%	13.35%	13.07%
Name	Actors	7	11.77%	11.44%	11.49%
Name	Author	18	11.35%	11.28%	11.10%
Name	Genre	20	10.67%	10.76%	10.96%
Author	Name	13	9.10%	9.10%	9.10%
Actors	Name	10	8.86%	8.86%	8.83%
Name	Tags	16	8.31%	8.57%	8.50%
Director	Name	14	5.35%	5.43%	5.58%
Tags	Name	9	4.00%	3.79%	4.14%
Name	Language	8	2.70%	2.79%	2.96%
Name	Rating	10	0.28%	0.00%	0.00%
Name	Votes	7	0.11%	0.00%	0.00%
Total		160	100.0 %	100.0 %	100.0 %

Table 7: Overview on Question Classes

4.2.3 Extensions to the Movie Dialog Dataset - QA task

This section proposes two extensions of the MovieDD-QA task. While the original MovieDD datasets provide question-answer pairs, training the model as discussed in Section 4.1.4 requires question-query pairs. Therefore, this section will present the annotation with intermediate labels. Furthermore, training neural architectures is known to be data intense. In order to compare the proposed training methods regarding their demand for data, this section introduces a smaller version of the training set.

4.2.3.1 Production of Intermediate Labels

Since the database operation $query(\cdot)$ breaks the differentiability of the model, question-answer pairs can not be used for supervised training. Intermediate labels represent the ground truth y to the sequence $\hat{y}$ that is input to the database operation $query(\cdot)$. As discussed in Section 4.2.2.1, the meaning of each question can be captured by the parameters QF, QE and RF. Due to the variable length of the QE, each question was annotated with a sequence of four words. While the first word represented the identified category of the QF, the fourth word represented the category of the RF. The second and third word of the label capture the first and the final word of the QE. This design implicitly assumes that all words in between the start and final word belong to the QE. Thereby, it is possible to describe QEs of different lengths with only two parameters.

This design enables the algorithm to deal with entities that consist of only one word. Given the question: "who acted in Boca?", the second and the third token of the label will refer to *Boca*. In a similar way, longer query entities can be handled. For example, in "James B. Clark was the director of which movies?", the QE consists of three words. In this case, the second token of the label refers to *James*, since it represents the start word of the QE, whereas the third token of the intermediate label refers to *Clark*, since it is the final word of the QE. When the actual query is produced, *B.* can be included since it is embraced by *James* and *Clark*.

The production of intermediate labels was eased by the fact that the MovieDD includes a list of all entities used throughout the different tasks. Thereby, it was possible to automatically extract the query entities for most of the question. As a result, the number of potential question patterns was reduced to about 500. These candidates included cases that where unambiguous and therefore led to incorrectly removed entities. For instance the question "what was the release date of the movie Magic Mike?", contains the entity *magic mike*. However, there is also a film called *movie magic*. Hence, the list of potential question patterns was reviewed by hand and compiled to 160 actual question patterns. Each of this question patterns was manually annotated with the ground truth sequence discussed previously. Afterwards, each question that groups into a certain question patterns was automatically annotated with the same intermediate label.

4.2.3.2 Introduction of a Reduced Training Set

In order to compare the proposed training approaches regarding their demand for data, this thesis extends the MovieDD-QA task by a smaller version of the training set. This dataset will be referred to as the reduced training set. In total, the reduced training set contains 10k examples, which were randomly sampled from the original dataset with respect to the frequency of question classes (see Table 7 in Chapter 4.2.2.2). Hence, the reduced training set shares the same characteristics as the original dataset.

4.3 Implementation Details

This section presents the implementation of the proposed Seq2Seq model into a dialog system that queries an Elasticsearch instance to answer natural language questions. Due to the production of actual queries, one can classify the KB interaction as hard-KB lookup. Figure 22 shows the implementation of both components into the system architecture. Given a natural language question the Seq2Seq model will predict a sequence that is used to fill three slots in the query interface. The resulting query is operated on an Elasticsearch instance to retrieve a database entry which answers the question.

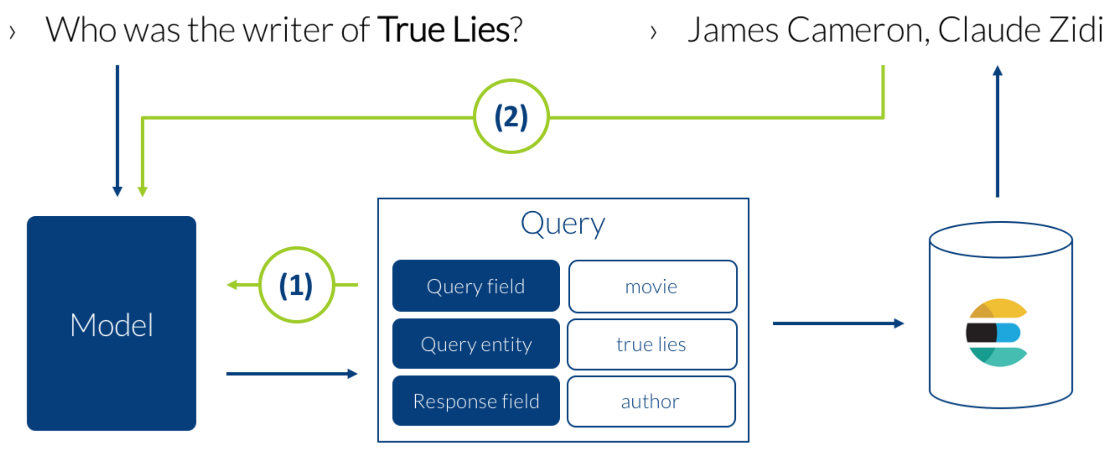


Figure 22: System Architecture

Training the model is possible in two ways. (1) Since the database operation breaks the differentiability of the system, intermediate labels, which represent the query ground truth, are used to train the model in a supervised fashion. (2) Leveraging the *REINFORCE* algorithm allows to overcome the problem of non-differentiability and one can train the model on question-answer pairs.

4.3.1 Model

In each training iteration, the model receives a lower case version of the inputs. This thesis defines the three subsets of the input x as the following. The subset x^c contains the 11 database categories, described by a single word. Furthermore, x^q represents a question sampled out of the dataset. Finally, the vocabulary of command tokens x^v only contains a start-of-sequence token. Since it does not contain an end-of-sequence token, the length of the output sequence $\hat{y}$ will be set to four. Let $\hat{y}_1$ represent the QF, then $\hat{y}_2, \hat{y}_3$ represent the first and final word of the QE, whereas $\hat{y}_4$ represents the RF.

Figure 23 illustrates the functionality of the model. As described in Chapter 4.1.1, the encoder is a two-layer bidirectional LSTM that reads the embedded inputs. This thesis uses 300-dimensional GloVe embeddings that were pre-trained on the 42B corpus. The final hidden states of the encoder are concatenated and passed to the decoder (Chapter 4.1.2). Such is a two-layer unidirectional LSTM, which takes the outputs from time step $t-1$ as inputs. At each decoder time step t the model attends over the encoder hidden states (Chapter 4.1.3) and computes a probability distribution over the input vocabulary. While training with intermediate labels, the most likely token from the input vocabulary is selected (Chapter 4.1.4). When the model is trained with policy gradient, the probability distribution is considered as a stochastic policy and predicting the next token in the output sequence is the corresponding action (Chapter 4.1.5). The hidden size of all LSTM-layers is defined as 100. Furthermore, dropout of 0.3 is used in encoder and decoder.

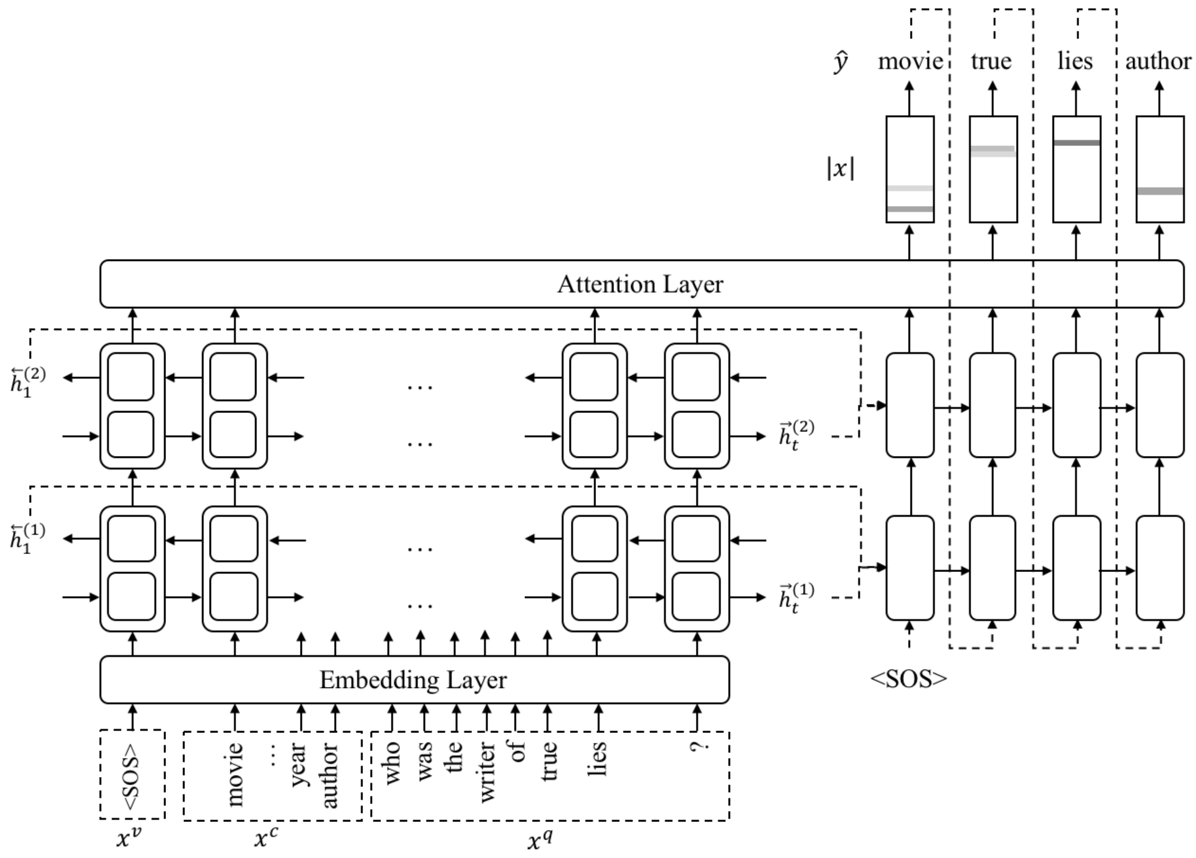


Figure 23: Seq2Seq Model

4.3.2 Query Interface

The interaction between the model and the Elasticsearch instance was implemented by the python libraries *Elasticsearch-py* and *Elasticsearch-dsl*. While *Elasticsearch-py* was used to set up an Elasticsearch client, *Elasticsearch-dsl* was used to query the database.

The query interface is used to implement the database operation $query(\cdot)$, which takes the models output sequence $\hat{y}$ as input. First and final word of the QE $(\hat{y}_2, \hat{y}_3)$ are used to extract the complete QE out of the question x^q. Then, the QF $(\hat{y}_1)$ and the QE are used to produce an Elasticsearch *term query*, which searches the index for exact matches. The RF $(\hat{y}_4)$ of the first retrieved document is used as response ω.

Instead of using term queries one could implement *match queries*. However, such would cause a credit assignment problem, since they are able to process full questions. Thereby, the boundaries between the capabilities of Elasticsearch and the proposed Seq2Seq model would blur.

4.3.3 Experiments

This section outlines the experiments that were conducted in this thesis. Since this thesis extended the MovieDD by intermediate labels, the quality of these annotations will be evaluated in a first step. Therefore, the database operation will be executed based on the intermediate labels. As a result, mismatches between ground truth question-query pairs and ground truth question-answer pairs can be identified.

Afterwards, this thesis investigates on the performance of the proposed Seq2Seq model. For all experiments, ADAM will be used as the optimizer and the learning rate is defined as 0.0001. Early stopping is applied if the accuracy on the validation set does not increase for a period of 15 epochs. Furthermore, the model will be trained on three different variants of the training set. Firstly, the original training set with 96k examples will be used to train the model. In a second line of experiments, the model is trained on the reduced training set with 10k examples. Finally, this thesis uses a version of the original training set, where two question patterns are removed from each question class. As a result, 81k examples remain in the dataset. This third version of the training set is applied in order to determine the capabilities of the model to generalize on unseen question patterns. In addition, one can distinguish the experiments with respect to the training method.

On the one hand, the proposed model is trained with cross entropy loss based on question-query pairs. For these experiments, the Mini-Batch Size (MBS) is set to 256. While training on the reduced training set (10k) was performed for 50 epochs, models were trained for 30 epochs on the original dataset (96k) and the generalization dataset (81k).

In contrast, all experiments that implement training with policy gradient on question-answer pairs are trained for 50 epochs. However, different MBS (128, 256 and 512) are determined. To limit the complexity of the problem and speed up learning, the pointer mechanism only attends over a reduced set of the inputs if trained with policy gradient. While predicting the QF and RF, the set of relevant inputs is defined as the database categories x^c. To predict the first and the final word of the QE, the pointer mechanism attends over the input question x^q. Furthermore, this thesis investigates on different configurations of the reward function. According to the reward function $R = [-2, -1, +z]$, the model receives a positive reward z if the execution of $query(\hat{y})$ yielded a correct result. During the experiments, the positive reward z will be varied (+1, +5, +25). Additional experiments investigate on the influence of a count-based exploration bonus. Therefore, the set of relevant time steps is defined as $T^{R^+} = \{1, 4\}$, to provide an incentive for the model to deviate from predicting the predominant category *name* for the QF and RF.

Each hyperparameter configuration is used in three experiments to determine the stability of the results. All experiments are run on a single graphics processing unit (GPU) of type Nvidia Titan X. Appendix A.2 provides a list of packages used in the implementation.

5 Results and Discussion

This chapter presents the results of the conducted experiments, which had a combined runtime of more than 400 hours. In order to provide appropriate evaluation tools, the following section will elaborate a set of metrics in a first step. In a second step, the quality of the intermediate labels will be determined. Third, an overview of the observed results when training the model with intermediate labels is reported. Afterwards, the discussion will turn to the outcomes of training the model with policy gradient. Finally, the performance of the proposed hard-KB lookup model will be compared to models that performed soft-KB lookup.

5.1 Evaluation Metrics

This section discusses how the system's performance will be evaluated. It will start by defining the general notation and finish by the deriving specific metrics from this notation.

The following parameters will be used to specify the quantity of examples in a certain split of the dataset, e.g. the test set. While N will denote the total number of examples in such split, N_{vq} refers to the number of produced valid queries. A query is considered a **valid query** if its operation against the database resulted in an non-empty response and caused no error. Furthermore, the number of executed queries that yielded a correct result is denoted as N_{ex}. If the ground truth of a question is ambiguous, e.g. an actor performed in multiple movies, a database response is assessed **correct result** if it matches with one of the ground truth items. According to the KB, the actor Willem Dafoe starred in 28 movies. As soon as the model predicts a query, which when executed returns one of these 28 films, it is a correct result. Finally, N_{qm} is defined as the number of examples where the string of the predicted query equals the string of the target query. This will also be referred to as **query match**.

Previous experiments on the MovieDD reported the *hits*@k metric. This metric considers the first k answers and checks if one of these results matches the ground truth. Dodge et al. [Dod+15] use *hits*@1 to evaluate the QA task of MovieDD. However in the case that $k = 1$ it equals the executional accuracy as proposed by Zhong, Xiong and Socher [ZXS17], who defined it as $\text{Acc}_{ex} = \frac{N_{ex}}{N}$. Furthermore, the query match accuracy $\text{Acc}_{qm} = \frac{N_{qm}}{N}$ refers to the share of examples where the predictions of the QF, QE and RF matched the target query. Finally, the share of valid queries is defined as $\text{Acc}_{vq} = \frac{N_{vq}}{N}$. This set of metrics will be used to evaluate the general performance of the model through different runs.

In addition, the predictions of the slots in the interface will be evaluated with another set of metrics. Let N_{qm}^{slot} be defined as the number of times the prediction of specific *slot* was correct. In case of the QF or RF this equals the amount of correctly predicted categories, whereas for the QE slot this number equals the amount of correctly predicted entities. Based on this count, the slot specific query match accuracy Acc_{qm}^{slot} can be computed by $\frac{N_{qm}^{slot}}{N}$.

Let entity e refer to the object of a question x^q, then e_1 represents the first word of the entity and e_{-1} represents the final word of the entity. The position of e_1 in the input sequence x^q will be defined as λ_{start}, whereas λ_{end} denotes the index of the final word of the entity e_{-1}. Given the question: "Who was the writer of True Lies?", the entity e is the movie *True Lies*, which ranges from position 6 to 7 in the input sequence. Hence λ_{start} is set to 6, whereas λ_{end} equals 7. Note that the definition of λ is unambiguous.

Thus, the length of an entity L can be defined as

$$L_j = \lambda_{end,j} - \lambda_{start,j}, \tag{5.1}$$

where j is an index to distinguish from other samples of the dataset. As the model only predicts the start and the final of the entity, it can appear, that it predicts $\lambda_{start} > \lambda_{end}$. Hence, entities with a negative length are referred to as **invalid entities**. They represent one of the reasons that may cause invalid queries. In order to represent the average entity length $\bar{L}$ all valid entities are considered. Furthermore, the share of invalid entities is denoted as κ.

To determine the deviation between predicted and ground truth entity, the distance d_{ij} is defined as

$$d_{ij} = |\lambda_{ij}^{y} - \lambda_{ij}^{\hat{y}}|, \tag{5.2}$$

where $i \in \{start, end\}$ indicates if it is about the first or the final word of the entity, while j is the index of the sample. Furthermore, $\lambda_{ij}^{\hat{y}}$ represents the predicted entity, whereas the ground truth entity position is denoted as λ_{ij}^{y}. Finally, this distance makes it possible to define a quality measure Δ_i with $i \in \{start, end\}$ as

$$\Delta_i = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{1 + d_{ij}}. \tag{5.3}$$

This quality measure is one, if the distance between predicted entity and ground truth is zero, which means that their start and end words are equal. In contrast it decreases if the distance rises.

Eventually, another metric for the evaluation of the QF and RF slot will be introduced. Let N_c^{QF} represent the number of times category c was predicted for the QF slot. Likewise, N_c^{RF} is defined as the count of category c in the RF slot. Based on these counts, the prior probability that the model selects category c to fill a respective slot is given by $p_c^{slot} = \frac{N_c^{slot}}{N}$. Typically, the Shannon Entropy is used to describe the uncertainty of decisions [GBC16]. However in this thesis it is used to identify whether the model tends to predict a predominating category with a high probability or a set of different categories with smaller individual probabilities. Therefore, the entropy $H(p)$ is defined as

$$H(p) = -\sum_{c \in \mathcal{C}} p_c \log_{\mathcal{C}}(p_c), \tag{5.4}$$

where p is the set of category probabilities p_c with $c \in \mathcal{C}$. Additionally, the logarithm to the base of $\mathcal{C}$ instead of log_2 is used to avoid that the entropy becomes larger than one. This transformation has no effect on the following features of the entropy. First, if only a single category is selected, the entropy yields zero. Second, if a set of categories is drawn from a uniform distribution, the entropy yields one. This thesis mainly aims for the former feature to identify when the model predicts only one predominant category.

5.2 Evaluation of the Annotated Dataset

The original dataset by Dodge et al. [Dod+15] consists of question-answer pairs. Since one version of the proposed model is trained on question-query pairs, a pre-processing step was necessary to annotate the data in order to produce intermediate labels. As the authors provided a list that contained all entities of the dataset, it was possible to create an automated approach to retrieve the different question patterns in the dataset. Each question pattern was annotated manually and used to add the intermediate label to each question in the dataset.

The following contemplation evaluates the quality of these intermediate labels. Table 8 reports the results across the different datasets. The overview includes the performance on the regular training set, which contains 96,185 questions, and on the reduced training set, which contains 10,000 questions. This reduced version was sampled from the regular one. In order to evaluate the performance of the annotated labels, the percentage of valid queries Acc_{vq} and the executional accuracy Acc_{ex} were observed.

Dataset	**Size**	**Acc_{vq}**	**Acc_{ex}**
Train	10,000	94.1%	90.8%
Train	96,185	94.1%	91.0%
Dev	9,968	94.1%	91.1%
Test	9,952	94.5%	91.2%

Table 8: Evaluation of the Target Labels

These observations show a uniform behavior across the splits of the dataset. About 94% of the produced labels are considered as a valid query, whereas about 91% of all queries yield a correct result. This leads to the conclusion, that a model trained on these labels can achieve an executional accuracy Acc_{ex} of approximately 91% at a maximum. Thus, the quality of the intermediate labels represent an important limitation to the actual performance of the supervised model. At this point, removing the corresponding question-query pairs from the dataset would be a valid step. As a result one would enable the model to learn on labels of higher quality. However, this step would influence the comparability of the results of this thesis. Dodge et al. [Dod+15] used the full dataset to evaluate different models performing soft-KB lookup.

To maintain full comparability, it was decided to keep the question-query pairs that lead to incorrect results in the dataset. Nevertheless, an investigation on the causalities behind this outcome was conducted. This analysis unveiled two reasons for this problem.

First, about 5% of the queries are invalid as they yield no response by the database. This is due to a design choice of the interface (discussed in Section 4.3.2) between the model and the Elasticsearch instance. It appeared, that this design is not able to process certain special characters. Dissolving this problem is difficult, as it would either result in an increased complexity of the interface or blur the boundaries between the capabilities of the Seq2Seq model and optimized retrieval methods of Elasticsearch.

The second reason that contributes to imperfect performance of the intermediate labels is based on ambiguities in the knowledge base. For instance, the training set contains the following question: "what was the release date of the film the three musketeers?". The original dataset labels this question with the answer 1939. Indeed the knowledge base includes a movie named *the three musketeers* released in 1939. Nevertheless, there are four other movies that share the same name. Those movies were released in 1973, 1993, 1994 and 2011 respectively. This example highlights the problem of ambiguities in the dataset. It causes about 1% of the invalid queries and the gap of about 3% between the share of valid queries Acc_{vq} and executional accuracy Acc_{ex}. Regarding the experiments of Dodge et al. [Dod+15] one can presume that they face the same problem with ambiguities.

The analysis of the produced intermediate labels indicate an approach based limitation. Reflecting the trade-off between the adjustment of this miss-behavior and resulting loss of comparability lead to the conclusion to accept imperfect labels but have full comparability. Furthermore it was shown, that ambiguities in the dataset cause correctly annotated questions to fail in execution.

5.3 Training with Intermediate Labels

This section evaluates the performance of the supervised variant of the proposed dialog system. As introduced in Chapter 4.1.4, it is trained on question-query pairs and not on the actual answer that was retrieved from the database. However, an end-to-end dialog system will be evaluated on the performance on question-answer pairs. Therefore, the following subsection provides an analysis that considers both aspects. Furthermore, it will be shown how the different slots of the interface are learned. Later on in this chapter, the generalization capability of the model will be discussed.

5.3.1 General Performance of the Supervised Model

In order to analyze the models performance on question-query pairs as well as the quality of the answers retrieved from the database, several metrics are used. First, the query match accuracy Acc_{qm}, which is related to the optimization while training on intermediate labels. Second, the valid query accuracy Acc_{vq}, which shows the share of produced valid queries. Finally, the executional accuracy Acc_{ex}, which shows the share of questions that were answered correctly. Table 9 presents the results that were obtained during our experiments.

Train Size	**Run**	**Acc$_{qm}$**	**Acc$_{vq}$**	**Acc$_{ex}$**
10k	1	97.3%	93.0%	89.2%
	2	96.9%	92.7%	88.8%
	3	97.3%	93.1%	89.4%
	∅	97.2%	92.9%	89.1%
96k	1	99.7%	93.9%	90.6%
	2	99.7%	93.9%	90.5%
	3	99.7%	93.9%	90.6%
	∅	99.7%	93.9%	90.6%

Table 9: Performance of the Model Trained with Intermediate Labels

It appears, that the dialog system performs well on both sizes of the training set and approaches the upper performance boundary of 91.2% executional accuracy, which was discussed in the previous section. On the larger training set it was observed, that the model learns fast and improves the quality of its prediction for all slots in the interface in every step. Training on this dataset was stopped after the query match accuracy exceeded 99.5% and the task was therefore considered as solved. For all three runs this happened to be between epoch 8 and 11. Models trained on the reduced dataset did not solve the task. However they achieved an average query match accuracy of 97.2%. Compared to the observations on the large training set, learning the different slots of the query interface went not straightforward. The performance on the reduced training set will be analyzed in the subsequent section.

5.3.2 Analysis of Learning on the Reduced Training Set

This section analyzes, how the performance developed during training on the reduced dataset. Therefore, the adjustments of the models predictive behavior were observed. Figure 24 shows the average performance observed while training the model on 10k examples. It illustrates the accuracy of the single slots: QF, QE and RF, as well as the correctness of the complete query. As one can see, it takes 9 epochs until the model produces queries that yield the correct result after they were operated. Between epoch 10 and epoch 20 the query match accuracy increases rapidly and after 20 epochs, the curve saturates.

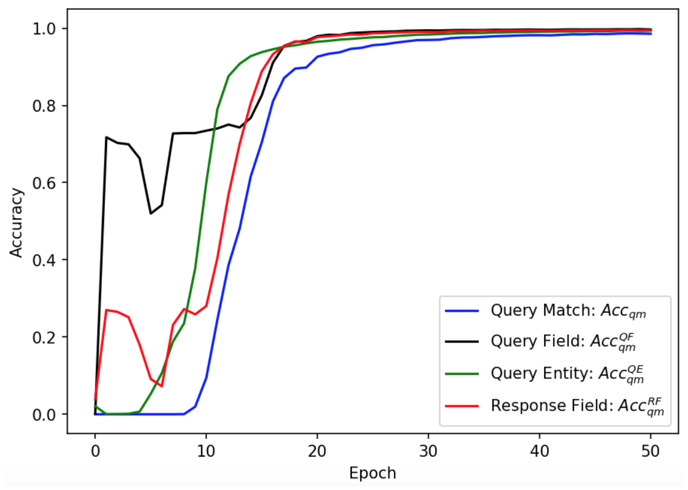
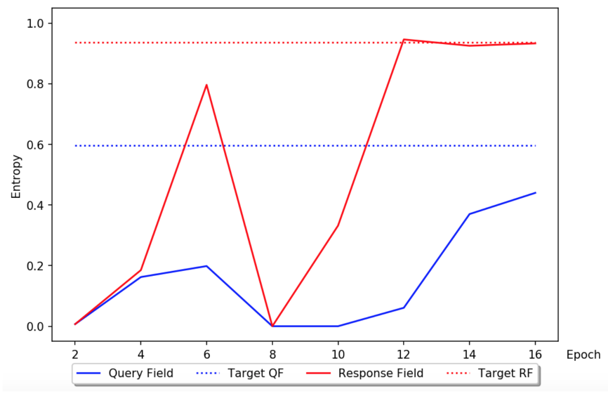


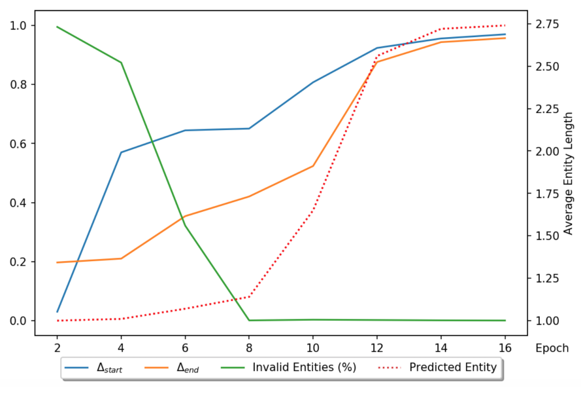
Figure 24: Learning Curve of the Supervised Model

Although the query match accuracy over time is a helpful indicator, it provides no information about the dynamics within the system. The further analysis will emphasize the adjustments in the initial phase of the experiments. Already after the first epoch, the accuracy of the QF and RF slot increased significantly. However, this is due to the fact, that the model selected the predominant category to fill these slots. Figure 25 (a) shows the entropy of the chosen categories to fill the QF, RF respectively. In addition to the entropy observed during training, the figure provides the entropy of the labels. As one can see, the entropy in Epoch 2 is nearly zero for both slots, which means that the model selects a single category for all samples in the epoch. Despite that, Figure 25 (b) presents different metrics for the QE slot. In Epoch 2, almost all entities that were predicted by the model were invalid and therefore hinder the database to retrieve a correct result.

After Epoch 4, one can observe a drop in Acc_{qm}^{QF} and Acc_{qm}^{QF}. The entropy in both slots increases from this epoch onwards, which means that the model deviated from choosing the predominant category and rather selected different ones. Later on, the model recovered and returned to this strategy in Epoch 8. Nevertheless, it appears that this through allowed the model to adjust towards the QE slot. The share of invalid queries decreased



(a) Entropy of the Query and Response Field



(b) Query Entity Specific Metrics

Figure 25: Evaluation of the Experiments on the Reduced Training Set

to almost zero in Epoch 8. Furthermore, the quality of the start word Δ_{start} and the end word Δ_{end} increased from 0.03, 0.20 respectively in Epoch 2 to 0.65, 0.42 respectively in Epoch 8.

Despite this improvement in the single slots, the model is still incapable to retrieve correct results from the database based on the produced queries. This is mainly caused by two reasons. First, as already discussed, the model predicts only the predominant category for the RF. This makes it impossible to retrieve the correct results for the rest of the questions, which account for about three quarters of the dataset. Second, as Figure 25 (b) shows, the average length of the predicted entity is 1.14, which does not seem sufficient to cover the complexity of the dataset. After Epoch 8, the observed metrics indicate a changing prediction behavior in all slots. While the entropy of the RF slot indicates that the distribution of predicted categories approaches a similar distribution as the target categories, the entropy of the QF increases more slowly. Both observations reflect the increase of the accuracy in both slots shown in Figure 24. Furthermore, one can see that the model predicts longer entities over time which enhanced the quality Δ_{start} and Δ_{end} to almost one. This suggests, that most of the entities were predicted correctly, which is confirmed by Acc_{qm}^{QE}.

One can conclude that the model trained supervised on the reduced dataset learns as follows. In the phase after initialization, the model tends to predict the predominant category for QF and RF. Furthermore, the majority of the predicted entities is invalid. Hence, the model is not able to retrieve correct results from the database. Around Epoch 6, the model adjusts its predictive behavior which leads to a drop in the accuracy of QF and RF. However, this adjustment enables the model to produce query entities of better quality. From this point onward, the share of invalid entities decreases, while the average length of the predicted entities increases. In addition, the model selects different categories than just the predominant one. The combination of these adjustments lead to an improved query match accuracy Acc_{qm}, which according to the definition of learning proposed in Chapter 2.2 indicates that the model actually learned from experience.

5.3.3 Generalization

In order to evaluate the capability of the model to generalize across unseen data, two question patterns of every question class were removed from the training set. The remaining training set consisted of 81k question-query pairs. This resulted into the fact, that during validation and testing, the algorithm needed to classify question patterns it had not seen in training. The share of the unseen data was about 14% for both sets. Table 10 provides the observations of the experiments conducted with a MBS of 256.

Train Size	**Run**	**Type**	**Acc_{qm}**	**Acc_{vq}**	**Acc_{ex}**
	1	Total	96.7%	91.6%	88.3%
81k	1	Patterns seen in train	99.0%	93.2%	90.0%
	1	Unknown patterns	82.4%	84.4%	78.4%
	2	Total	95.7%	90.6%	87.4%
81k	2	Patterns seen in train	99.0%	93.2%	90.0%
	2	Unknown patterns	75.7%	74.7%	71.8%
	3	Total	96.1%	91.6%	87.3%
81k	3	Patterns seen in train	99.1%	93.2%	90.0%
	3	Unknown patterns	79.0%	81.9%	74.9%
	∅	Total	96.2%	91.3%	87.9%
81k	∅	Patterns seen in train	99.0%	93.2%	90.0%
	∅	Unknown patterns	79.0%	80.3%	75.0%

Table 10: Generalization of the Supervised Model

It was found that removing 15.6% of the question patterns during training reduced the executional performance on the complete test set by 2.9% compared to the results presented in Section 5.3.1. As expected, there was no difference for question patterns the algorithm already was familiar with. However, the executional performance on question patterns that were removed during training decreased to 75.0%. A similar drop was observed in case of the query match accuracy, which fell from 99.7% to 79.0%, and the share of valid queries, which was reduced to 80.3%.

Further investigation on the mis-classifications revealed a reduced performance in the prediction of the QE and the RF for unknown question patterns. Observations of the predictive behavior in the QE slot showed the following. In most of the cases, the model predicts the first word of the entity correctly, which corresponds to a quality Δ_{start} of 0.97. As depicted in Figure 26, entities with a length of four words or more are predicted more often than they actually appear in the ground truth. This means in some cases, the model includes additional words into the prediction of the QE, which increases the average length of the predicted entities $\bar{L}_{pred}$ to 2.89 ($\bar{L}_{target} = 2.61$) and reduces the quality of the last word of the entity Δ_{end} to 0.86. The execution of such queries will most likely retrieve empty answers, since a response must contain all words of the QE. Therefore, one can conclude that the adjusted behavior in the QE slot contributes to the drop in all three performance metrics (Acc_{qm}, Acc_{vq} and Acc_{ex}).

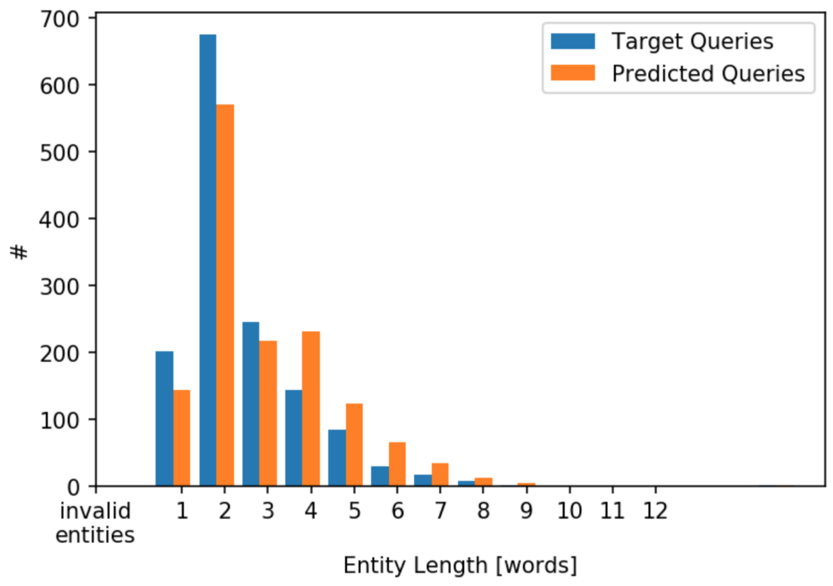


Figure 26: Distribution of the Entity Lengths in Run 2 (Unknown Patterns)

Moreover, the performance of the RF Acc_{qm}^{RF} dropped from 99.4%, for the patterns seen in training, to 89.7%, for the patterns not seen in training. This drop is mainly caused by samples which have *actor* as their RF ground truth. In this cases, it was observed that the model tended to predict *author* as the RF. It appears that there are only seven question patterns which address the RF *actor* and two of them were removed to test the models capability to generalize.

As about 75.0% of the knowledge base entries include an author, most of the produced query will retrieve an answer and are therefore considered a valid query. This means, that the faulty predictive behavior will mainly cause Acc_{qm} and Acc_{ex} to drop.

5.4 Training with Policy Gradient

Training the model architecture with policy gradient is based on the question-answer pairs of the original dataset. This section will outline several observations that were obtained during training. First, the influence of the reward function will be discussed. It was found that the design used by Zhong, Xiong and Socher [ZXS17] was not sufficient to foster learning. The adjustment of the reward function lead to an improved performance. Later on, the effect of introducing a exploration bonus for rarely selected actions is evaluated. Finally, the capability of the model to generalize on unseen data will be discussed.

5.4.1 The Design of the Reward Function Matters

In this chapter, the outcome of the application of different reward functions is discussed. Throughout this chapter the exploration bonus will be inactive. The initial experiments in this thesis were conducted with the same reward function as Seq2SQL [ZXS17], which is defined as: $R = [-2, -1, +1]$. However, it was experienced that in this setting the model did not learn to produce queries, which retrieve correct results when executed against the Elasticsearch. Therefore, this section will start by discussing the difference between the experiments in this thesis and the setting of Zhong, Xiong and Socher [ZXS17]. Afterwards the obtained results will be presented.

While this work uses the MovieDD, Seq2SQL is trained on the WikiSQL dataset. Such is an open-domain dataset, which provides both, question-query and question-answer pairs. The size of the WikiSQL is comparable to the MovieDD. Although, the number of categories in WikiSQL is significantly larger. The dataset is designed to test the generalization across predicted categories. Zhong, Xiong and Socher [ZXS17] trained their model for 300 epochs, whereas this implementation uses 50 epochs as the standard training interval. This choice is an arbitrary decision and based on runtime issues. Furthermore, they implemented their model with 200 units at each hidden layer, while this model consists of 100 units. Finally, they use a network-architecture similar to the one proposed in this thesis as one out of three components for their Seq2SQL model, which makes it more complex and powerful. Based on their experiments, they report, that training their Seq2SQL by policy gradient improves the performance by a small margin [ZXS17].

In contrast the experiments conducted in this thesis show that a model trained with the reward function $R = [-2, -1, +1]$ is not able to retrieve correct results. Table 11 provides an overview of 18 runs, which where performed on the reduced training set with different MBSs (128, 256) and three different reward functions. In addition, the table shows another 6 runs that were performed on the original training set. It appears, that the reward functions $R = [-2, -1, +1]$ and $R = [-2, -1, +5]$ enable the model to produce valid queries but queries yield poor results when executed.

In case of these reward functions, the conducted experiments indicate that the model acts myopic and focuses on short-term rewards. As the production of a valid query leads to a reward of -1 instead of -2, it seems possible that the model gets trapped in local optima, if it produces about 90% or more valid queries. Further analysis of the corresponding runs revealed the following. Across all runs, the model keeps a very simple schema to produce

			MBS 128		**MBS 256**	
Train Size	**Rewards**	**Run**	**Acc$_{vq}$**	**Acc$_{ex}$**	**Acc$_{vq}$**	**Acc$_{ex}$**
10k	[-2, -1, +1]	1	99.1%[1]	0.0%[1]	97.1%	0.0%
		2	97.8%	0.1%	92.2%	0.0%
		3	99.3%	0.0%	94.7%	0.0%
		∅	98.7%	0.0%	94.7%	0.0%
10k	[-2, -1, +5]	1	97.9%	0.0%	95.8%	0.0%
		2	98.3%	0.0%	96.6%	0.0%
		3	96.0%	0.0%	88.1%	0.1%
		∅	97.4%	0.0%	93.5%	0.0%
10k	[-2, -1, +25]	1	64.1%[1]	17.7%[1]	62.9%	29.2%
		2	62.9%	17.0%	64.1%	18.0%
		3	63.6%	17.4%	65.4%	45.8%
		∅	63.5%	17.4%	64.1%	31.0%
96k	[-2, -1, +25]	1	89.4%	25.4%	90.1%	68.6%
		2	88.4%	34.6%	88.5%[1]	47.4%[1]
		3	86.2%	47.5%	71.4%	47.5%
		∅	88.0%	35.8%	83.3%	54.5%

Table 11: Evaluation of Different Reward Functions

queries and does not adjust on different question patterns. Except for a single run, the entropy of the QF $H^{QF}(p)$ and the entropy of the RF $H^{RF}(p)$ where smaller than 0.05. This means, that with a few exceptions, the model uses a single category to predict the corresponding slot. Appendix A.3 provides a detailed overview of the figures used in this discussion. Furthermore, it contains an analysis of the first run with the reward function $R = [-2, -1, +1]$ and MBS 128, which showed a QF entropy of 0.33. In addition to the fact that the model predicts almost always a single category, it was found that it also kept the complexity of the QE at very low level. This is indicated by the quality measures Δ_{start} and Δ_{end}, which were 0.29 and 0.032 at a maximum, and 1.08 was the largest observed average query length $\bar{L}$, which was found. Despite this simplistic predictions it is worth noting that the share of invalid entities is at a very low level all the time. This supports the conclusion that the model optimizes towards valid query production.

While Table 11 only shows the results on the reduced training set for the reward functions $R = [-2, -1, +1]$ and $R = [-2, -1, +5]$, the same behavior was observed for $R = [-2, -1, +1]$ on the large training set (96k). The reward setting $R = [-2, -1, +5]$ was not tested on the original dataset. Instead another reward function $R = [-2, -1, +25]$ was considered. The consequences of this adjustment will be discussed subsequently.

[1]Appendix A.3 provides additional details about these runs.

First, the change of the reward function had an impact on the prediction of the QE. The quality metrics Δ_{start} and Δ_{end} increased significantly. A quality Δ of 0.5 would indicate that the index of the prediction differs in average one position from the target index. At this point it is worth noting that all configurations exceeded this threshold. Furthermore, the average entity length $\bar{L}_{pred}$ nearly doubled to more than 2. It appears that the model captures the actual entities contained in the question. This is confirmed by an increased query match accuracy of the QE Acc_{qm}^{QE}, which is above 50% for all observations. Given the initial reward function, this metric did not exceed 2%.

Second, it was found that the models predictive behavior for the RF changed. While the prediction was previously dominated by a single category, an enlarged entropy $H^{RF}(p)$ suggests that its now a balanced distribution over several categories. As a result, the accuracy of the RF Acc_{qm}^{RF} grew. However, it was observed that different configurations corresponded to different level of Acc_{qm}^{RF}. On both sizes of the training set, the MBS 256 achieved higher values than MBS 128. This reflects the hierarchy one can observe on the executional accuracy in Table 11.

Third, while predicting the QF, the model sticked to the behavior of choosing a single category over all samples. The QF entropy $H^{QF}(p)$ was zero in all but one run. Fourth, the share of produced valid queries Acc_{vq} dropped while training the model with the new reward function on the reduced training set.

Finally, the improved predictions of the QE and the RF increase the performance of the whole dialog system. It is now able to produce queries that retrieve correct answers from the database. However, the model is still limited in its capabilities. On the reduced dataset it achieves a Acc_{ex} of 45.8% at a maximum. Furthermore, it was found, that the executional accuracy varied by a significant margin during the different runs in the different configurations. Only the performance of the model using a MBS of 128 on the reduced training set seemed to be stable.

In conclusion, the conducted experiments indicate that the performance of the model depends on the scale of the rewards. While the impact on the predictive behavior of the model was analyzed, this section provided no discussion about possible reasons of this phenomenon. It appears that the feedback signal provided by low positive rewards, which are given for correct results, are not sufficient for the model to overcome local optima. On the algorithm side, the rewards influence the weight updates due to the expected return. According to Arulkumaran et al. [Aru^{+}17] this one is computed by averaging over trajectories, which are the samples in the mini-batch. It seems like this averaging over the batch size might cause the vanishing feedback signal in case of sparse positive rewards. Indeed, there is neither empirical nor theoretical proof for this contemplation. Therefore, further investigations are necessary to confirm the observed results and undertake the theoretical reasoning.

5.4.2 Exploration with Boni

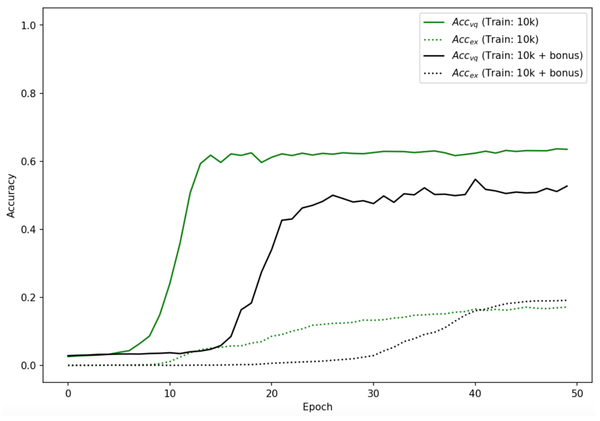
The previous section identified that the model tends to predict always a single category for the QF. Regarding the RF there is more variation in the performed actions. It appears that a deviation from this strategy is not fostered by the design of the proposed MDP. However, it is a desired behavior of the agent to not only select one category for the QF. Accordingly, an adjustment of the MDP was conducted. This included the introduction of a count-based exploration bonus as an additional component to the reward function. In Table 12 one can see the effect of the bonus for a MBS of 128 and 256.

			MBS 128		**MBS 256**	
Train Size	**Bonus**	**Run**	**Acc$_{vq}$**	**Acc$_{ex}$**	**Acc$_{vq}$**	**Acc$_{ex}$**
96k	False	1	89.4%	25.4%	90.1%	68.6%
		2	88.4%	34.6%	88.5%	47.4%
		3	86.2%	47.5%	71.4%	47.5%
		∅	88.0%	35.8%	83.3%	54.5%
96k	True	1	81.4%	70.8%	93.3%	75.2%
		2	96.4%	90.8%	92.0%	81.3%
		3	96.2%	91.0%	95.9%	90.7%
		∅	91.3%	84.2%	93.7%	82.4%

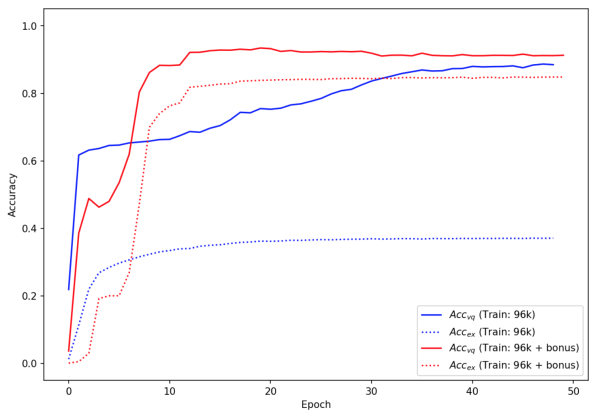
Table 12: Introducing an Exploration Bonus Improves Performance

Adding a count-based bonus for rarely executed actions yields a sharp increase in performance. Given a MBS of 128 training with a regular reward function achieved 35.8% executional accuracy in average, the system with boni achieved 84.2% in average. In Run 2 and 3 (with boni) the model approaches the best possible performance for the supervised model. With a MBS of 256 the performance of training without an exploration bonus is higher than the one of runs with MBS 128. After the introduction of the exploration bonus both MBSs perform on the same level. Furthermore, one can observe that Acc$_{vq}$ slightly increased for both configurations.

Figure 27 shows the influence of the exploration bonus on the average performance of the reduced training set and the original training set. As one can see, it causes counter intuitive results on the reduced training set. Even after 50 epochs of training, the valid query accuracy Acc$_{vq}$ for runs that used an exploration bonus is significantly lower than for runs without an exploration bonus. Furthermore, it takes about 30 epochs for the model trained with an exploration bonus to produce a significant amount of queries that retrieve a correct result. After the models terminate the executional accuracy Acc$_{ex}$ of the runs with an exploration bonus is slighlty higher than the ones of the runs without a bonus.



(a) Reduced Training Set (10k)



(b) Original Training Set (96k)

Figure 27: Evaluation of the Experiments with Mini-Batch Size 128

On the right hand side of Figure 27 one can see, that the exploration bonus slows down the improvement of the performance. Until epoch 6, the models trained without a bonus produce more valid queries and retrieve more correct responses. After this point, both metrics increase significantly for the models trained with an exploration bonus. After around 12 epochs one can observe that the curves of all metrics despite the valid query accuracy of the models trained without bonus saturate.

5.4.3 Generalization

This section investigates on the generalization capabilities of the model trained with policy gradient. Like for the evaluation of the supervised model (Chapter 5.3.3), about 15k samples were removed during training time.

It was found that the model trained with policy gradient also generalizes across unseen question patterns. Table 13 shows the obtained results. In each of the three runs, the model achieves an executional accuracy of more than 60% on the unseen data. This is less compared to the generalization of the supervised model. However, the models trained with policy gradient achieve a higher share of valid queries on the unseen data compared to training on intermediate labels. This is due to the fact that the reward function incorporates a feedback signal, which values valid queries over invalid queries. While Acc_{vq} shows no differences between patterns seen in training and unknown patterns, there is a is a gap of 12% to 14% in Acc_{ex}. As discussed in 5.3.3 this drop is influenced by the removal of samples which have *actor* as their RF ground truth.

Train Size	Run	Type	Acc_{vq}	Acc_{ex}
81k	1	Total	94.4%	87.3%
	1	Patterns seen in train	94.9%	89.3%
	1	Unknown patterns	91.5%	75.6%
81k	2	Total	91.3%	78.5%
	2	Patterns seen in train	91.5%	80.0%
	2	Unknown patterns	89.5%	68.7%
81k	3	Total	88.9%	74.4%
	3	Patterns seen in train	89.9%	76.3%
	3	Unknown patterns	83.3%	62.6%
81k	∅	Total	91.5%	80.1%
	∅	Patterns seen in train	92.1%	81.9%
	∅	Unknown patterns	91.4%	69.0%

Table 13: Generalization of the RL Model

5.5 Comparison to Soft-KB lookup

This section provides a comparison of the results found in this thesis and the ones reported by Dodge et al. [Dod+15]. While the model implemented in this study interacted with the KB via hard-KB lookup, Dodge et al. used soft-KB lookup.

Table 14 shows the combined results. Dodge et al. implemented six different models on the Movie Dialog Dataset. The performance of an LSTM, supervised embeddings and an ensemble of supervised embeddings is not competitive. For a discussion of these observations see Dodge et al. [Dod+15].

Methods	Lookup Type	Acc_{ex}
QA system [BCW14; Dod+15]	Soft	90.7%
LSTM [Dod+15]	Soft	6.5%
Supervised embeddings [Dod+15]	Soft	50.9%
MemN2N [Dod+15]	Soft	79.3%
Joint supervised embeddings [Dod+15]	Soft	43.6%
Joint MemN2N [Dod+15]	Soft	83.5%
This thesis (intermediate labels)	Hard	90.6%
This thesis (policy gradient)	Hard	84.2%

Table 14: Soft-KB Lookup vs. Hard-KB Lookup

As one can see, the model trained with policy gradient performs on a competitive level to the ensemble of Memory Networks and slightly better than a single Memory Network. An analysis on the differences in runtime and demand on resources would require further experiments since Dodge et al. [Dod+15] do not report such details. Since their model uses an attention mechanism over all entries stored in the KB it is likely to improve the runtime of their approach by utilizing additional computational power (GPUs). The bottleneck of the model trained with policy gradient is the interaction with the Elasticsearch instance. An enlarged computational power does not guarantee to speed up this interaction. Instead, one should focus on improving the interaction between the model and the Elasticsearch instance. Parallelizing the execution of multiple queries would require to set up multiple Elasticsearch instances storing the same knowledge base. However, this would lead to an overhead due to coordination tasks.

Despite, one should investigate if training an ensemble of the proposed Seq2Seq models can improve the overall performance. Regarding the observations of Dodge et al. [Dod+15] training an ensemble of models lead to an improvement in case of Memory Networks. However, training an ensemble of supervised embeddings reduced the performance by 7.3%.

Both, the ensemble of Memory Networks and the model trained with policy gradient are outperformed by the baseline reported in Dodge et al. [Dod+15]. Such is a standard QA benchmark which learns embeddings that match questions with database entries and was proposed by Bordes, Chopra and Weston [BCW14]. The gap between the performance of the model presented in this thesis and the QA baseline is 6.4%. Leveraging intermediate labels produced by human interaction enables the proposed model to approach the QA baseline and outperform all other methods presented by Dodge et al. [Dod+15].

The experiments conducted in this thesis show that interaction with a KB can be achieved by soft-KB lookup and hard-KB lookup on a competitive level. It appears, that both approaches tend to have certain advantages over the other one. Therefore, it depends on the use case which one should apply.

6 Conclusion and Outlook

This master thesis dealt with a problem setting in the domain of end-to-end dialog systems. Therefore, this thesis presented the state-of-the-art in dialog tasks. Since a vast amount of today's informational resources is stored in database, the interaction with a knowledge base was included into design of the system architecture. Typically there are two different ways how to interact with a knowledge base. Models can either perform soft-KB lookup, which means that they compute a probability distribution over a tabular knowledge base and detect the most important entry, or models perform hard-KB lookup, which means that they produce actual database queries. This thesis aimed for the latter case. However, this approach shows the problem that database operations are non-differentiable which impedes the training process of the model.

This thesis examined the capabilities of a state-of-the-art Seq2Seq model on a large scale question answering dataset, the Movie Dialog Dataset [Dod+15]. Thereby, two ways of training the model were investigated. On the one hand, a supervised training method based on intermediate labels was determined. This approach has two major shortcomings. Firstly, it requires human interaction to produce intermediate labels, which can be very cost intensive. Secondly, this approach does not consider the results that are obtained after a query was executed. During the conducted experiments, this model outperformed soft-KB lookup methods reported by Dodge et al. [Dod+15]. Furthermore, it yields competitive results to a specialized QA system [BCW14]. This leads to the conclusion that training on intermediate labels is a reliable option as long as it is possible to produce high quality labels with minimal effort and reduced human interaction. However, it appears that this limits such systems to the application in narrow domains.

On the other hand, this thesis applied the *REINFORCE* algorithm to train the Seq2Seq model. Since this approach takes natural language question as inputs and is trained on the database response after the execution of a query, it does not require intermediate human interaction. However, it was found that the design of the underlying MDP is crucial for the performance of the model. It was observed that the model tended to produce simplistic question schema, since it got stuck in local optima of the reward function. The introduction of a count-based exploration bonus solved this problem and caused a significant boost in performance. Although, it appears that training a model with deep reinforcement learning causes instabilities. For instance training the model with an exploration bonus resulted in a gap of 20% between two runs with the same configuration. Nevertheless, the average performance of training the proposed architecture with policy gradient was on the same level as the Joint MemN2N model by Dodge et al. [Dod+15]. The QA system [BCW14] and the supervised approach of this thesis performed better. For a more detailed analysis of the model's stability, further experiments need to be conducted.

Comparing the results of the two ways to train the model leads to two additional conclusions. Firstly, deep reinforcement learning is more data demanding than supervised training. Training the model with intermediate labels achieved satisfying results on the original training set (96k) and on the reduced training set (10k), whereas the training with policy gradient was only successful on the original training set. Secondly, the application of a state-of-the-art Seq2Seq architecture enables the model to generalize on unseen question patterns. The influence on the way to train the model on its capability to generalize is negligible.

Eventually, the conducted experiments revealed two reasons why one might prefer soft-KB lookup over hard-KB lookup. Since training the model with policy gradient requires an interaction with the database it is computational expensive. Running the original dataset on a single GPU took up to 12 hours of training, while training with intermediate labels on the same dataset only took 1 hour. This runtime issue represents a limitation for models that perform hard-KB lookup in general and for the experiments conducted in this thesis in particular. Furthermore, it was found that the interface used to connect the components of the system architecture, caused about 5% of the intermediate labels to be assessed as invalid queries. The analysis of this problem revealed the relationship to an important design choice. However, adjusting this design choice was not possible, as it would have blurred the line between the capabilities of the proposed Seq2Seq model and the capabilities of Elasticsearch. This problem is known as the credit assignment problem. This finding is not limited to this thesis, but includes all kind of dialog agents that perform hard-KB lookup. Systems, which perform soft-KB lookup do not face this problem, as they access the knowledge base directly.

Despite this final remarks the obtained results are promising for the following reasons. Since a large share of information is already stored in databases, it is more likely to access the existing structure with hard-KB lookup instead of redefining them into tabular forms. While this seems possible for relational databases like SQL, it is nearly impossible for non-relation databases. Furthermore, the potential of hard-KB lookup is not exhausted. Regarding an interaction with Elasticsearch further research might focus on utilizing the score of a retrieved answer. In this way one could reduce the uncertainty which is caused by the non-differentiable database operation. Given the retrieved top response has a low score, it is likely that the query is not sufficient to search for the ground truth. In contrast, if there are multiple response with a score above a certain threshold, it seems reasonable to introduce a rank-based reward to weaken the influence of ambiguities like the movie the three musketeers. In this case, the model would receive a reduced positive reward if the corresponding movie is ranked third or fourth instead of a negative reward.

One can conclude that there is still a long way until humans will interact with fully functional dialog systems. Regarding the experiments conducted in this thesis one can think of two directions of subsequent steps. On the one hand, one could investigate on the models performance on other tasks and corpora. This includes both ways of training the model and would allow a comparison to other models. On the other side, one could use the proposed architecture in order to design a more advanced model. This thesis relied on *REINFORCE* without a baseline, the next logical step would be to incorporate baselines or to implement an actor-critic approach. Regarding the discussions in Section 2.3.3 such might stabilize of the results. In addition, another exploration method or a different kind of embedding offers potential for improvement. Finally one can think of resolving the single turn interaction structure and test the architecture on interactions with multiple turns. Most likely this would require a memory module to that the model can write and attend.

A Appendix

A.1 Excerpt of the Proof of the Policy Gradient Theorem

$$\begin{aligned}
\nabla J(\theta) &= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla_\theta \pi(a|S_t, \theta) \right] & \times \frac{\pi(a|S_t,\theta)}{\pi(a|S_t,\theta)} \\
&= \mathbb{E}_\pi \left[\sum_a \pi(a|S_t,\theta) q_\pi(S_t, a) \frac{\nabla_\theta \pi(a|S_t,\theta)}{\pi(a|S_t,\theta)} \right] & A_t \sim \pi_\theta \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \right] & \nabla \log x = \frac{\nabla x}{x} \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \nabla_\theta \log \pi(A_t|S_t,\theta) \right] & \mathbb{E}_\pi \big[G_t | S_t, A_t \big] = q_\pi(S_t, A_t) \\
&= \mathbb{E}_\pi \left[G_t \nabla_\theta \log \pi(A_t|S_t,\theta) \right]
\end{aligned} \tag{A.1}$$

A.2 Implementation Details

This thesis is implemented in Python (Version 2.7.14). The following list represents the main packages that were used for implementation and visualization in alphabetic order:

- Elasticsearch-dsl (v6.1.0)
- Elasticsearch-py (v6.1.1)
- Matplotlib (v2.1.1)
- Numpy (v1.13.3)
- Pytorch (v0.3.0)
- Scikit-learn (v0.19.1)

A.3 The Design of the Reward Function Matters

This appendix provides additional details about Section 5.4.1

A.3.1 Average Metrics

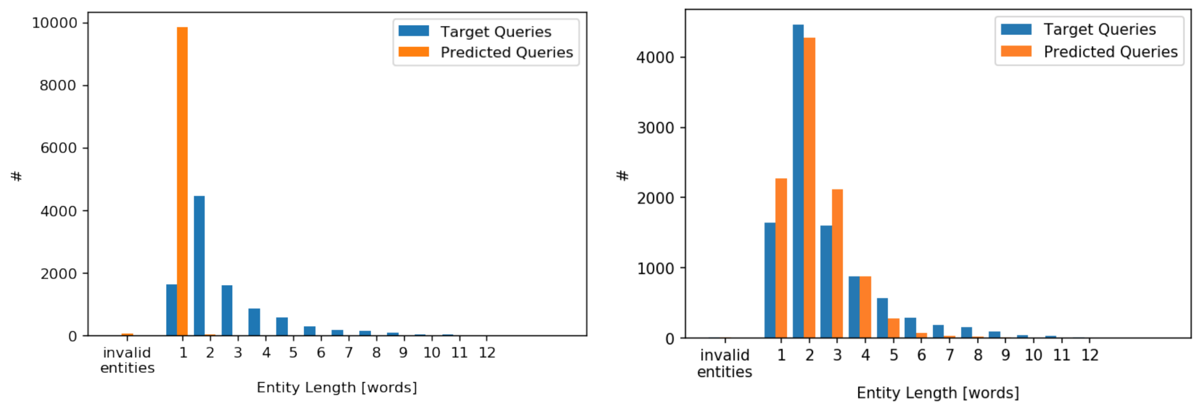
Table 15 provides an overview of the average metrics obtained in the experiments.

Train Size	Rewards	MBS	Δ_{start}	Δ_{end}	$\bar{L}_{pred}$	κ	$H^{QF}(p)$	$H^{RF}(p)$
10k	[-2, -1, +1]	128	0.170	0.005	1.01	0.7%	0.120	0.048
		256	0.198	0.015	1.05	3.1%	0.011	0.017
10k	[-2, -1, +5]	128	0.166	0.005	1.03	1.4%	0.0	0.001
		256	0.218	0.022	1.06	3.0%	0.013	0.018
10k	[-2, -1, +25]	128	0.73	0.89	2.3	0.3%	0.0	0.760
		256	0.67	0.83	2.2	0.3%	0.0	0.753
96k	[-2, -1, +25]	128	0.53	0.73	2.0	0.2%	0.0	0.579
		256	0.71	0.83	2.1	0.4%	0.146	0.875

Table 15: Average Metrics on Different Configurations

A.3.2 Analysis of Different Runs

Table 16 summarizes the hyperparameters and results of the runs and Figure 28 to 30 illustrate the distributions of average entity length and the confusion matrices respectively.



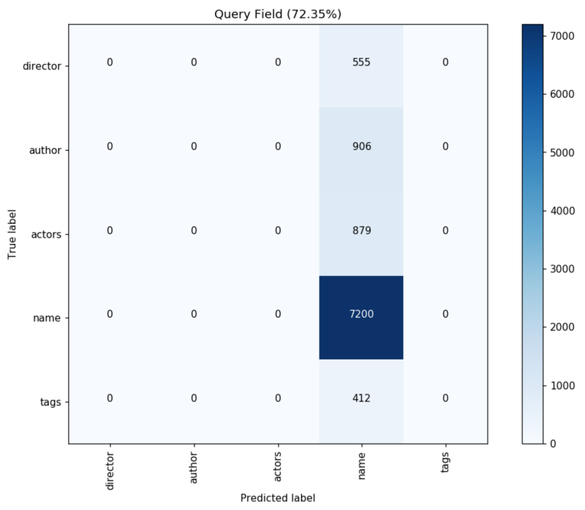
(a) The model develops a simple schema to produce valid queries (A)

(b) The model learns to retrieve the entities out of the questions (B)

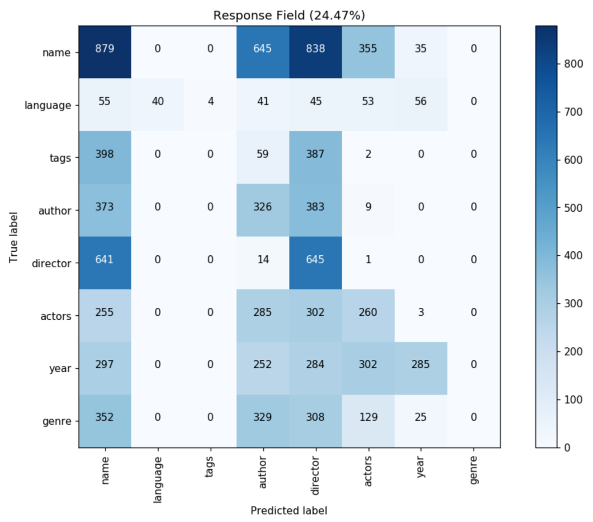
Figure 28: Distribution of Entity Lengths

	A	**B**	**C**
Train Size	10k	10k	96k
Rewards	[-2, -1, +1]	[-2, -1, +25]	[-2, -1, +25]
Run	1	1	2
MBS	128	128	256
Bonus	False	False	False
Acc_{vq}	99.1%	64.1%	88.5%
Acc_{ex}	0.0%	17.7%	47.4%
Acc_{qm}^{QF}	4.3%	72.3%	72.3%
Acc_{qm}^{QE}	0.0 %	68.3%	51.8%
Acc_{qm}^{RF}	0.5%	24.5%	61.4%
Δ_{start}	0.162	0.77	0.64
Δ_{end}	0.001	0.9	0.7
$\bar{L}_{target}$	2.78	2.78	2.78
$\bar{L}_{pred}$	1.01	2.3	1.94
κ	0.5%	0.5%	0.4%
$H^{QF}(p)$	0.333	0.0	0.0
$H^{RF}(p)$	0.128	0.745	0.861

Table 16: Summary

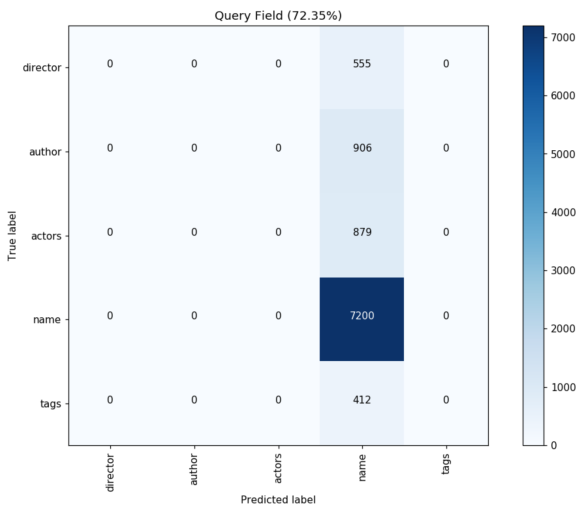


(a) Query Field

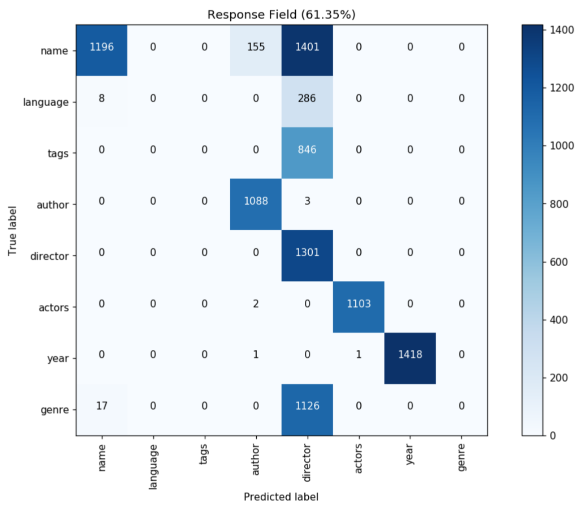


(b) Response Field

Figure 29: Confusion Matrices of Query and Response Field (B)



(a) Query Field



(b) Response Field

Figure 30: Confusion Matrices of Query and Response Field (C)

References

[Ama18] Amazon. *What is Elasticsearch?* 2018. URL: https://aws.amazon.com/elastic-search-service/what-is-elasticsearch/ (visited on 05/24/2018).

[Arg+09] Brenna D. Argall et al. “A survey of robot learning from demonstration”. In: *Robot. Auton. Syst.* 57.5 (2009), pp. 469–483. arXiv: 1105.1186v1.

[Aru+17] Kai Arulkumaran et al. “A Brief Survey of Deep Reinforcement Learning”. In: (2017), pp. 1–16. arXiv: 1708.05866.

[BBW17] Antoine Bordes, Y-Lan Boureau, and Jason Weston. “Learning End-to-End Goal-Oriented Dialog”. In: (2017), pp. 1–15. arXiv: 1605.07683.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: (2014), pp. 1–15. arXiv: 1409.0473.

[BCW14] Antoine Bordes, Sumit Chopra, and Jason Weston. “Question Answering with Subgraph Embeddings”. In: (2014). arXiv: 1406.3676.

[Ben+03] Yoshua Bengio et al. “A Neural Probabilistic Language Model”. In: *The Journal of Machine Learning Research* 3 (2003), pp. 1137–1155. arXiv: 1301.3781v3.

[Ber+13] Jonathan Berant et al. “Semantic Parsing on Freebase from Question-Answer Pairs”. In: *Proceedings of EMNLP* October (2013), pp. 1533–1544. URL: https://www.aclweb.org/anthology/D/D13/D13-1160.pdf.

[Bor+15] Antoine Bordes et al. “Large-scale Simple Question Answering with Memory Networks”. In: (2015). arXiv: 1506.02075.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning Long-Term Dependencies with Gradient Descent is Difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. arXiv: 1211.5063v2.

[Cha+17] Francois Chaubard et al. *Natural Language Processing with Deep Learning.* 2017. URL: https://web.stanford.edu/class/cs224n/lecture{_}notes/cs224n-2017-notes1.pdf (visited on 05/23/2018).

[Che+17] Hongshen Chen et al. “A Survey on Dialogue Systems: Recent Advances and New Frontiers”. In: 1 (2017). arXiv: 1711.01731.

[CHH02] Murray Campbell, a. Joseph Hoane Jr., and Feng-hsiung Hsu. “Deep Blue”. In: *Artificial Intelligence* 134.1-2 (2002), pp. 57–83. URL: http://www.sciencedirect.com/science/article/pii/S0004370201001291.

[Cho+14] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: (2014). arXiv: 1406.1078.

[Cod70] Edgar F. Codd. “A Relational Model of Data for Large Shared Data Banks”. In: *Communications of the ACM* 13.6 (1970), pp. 377–387. arXiv: `1011.1669v3`.

[Col81] Kenneth Mark Colby. “Modeling a paranoid mind”. In: *Behavioral and Brain Sciences* 4.04 (1981), p. 515.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. In: (2015), pp. 1–14. arXiv: `1511.07289`.

[CW08] Ronan Collobert and Jason Weston. “A unified architecture for natural language processing”. In: *Proceedings of the 25th international conference on Machine learning - ICML ’08* (2008), pp. 160–167. arXiv: `1603.06111`.

[Cyb89] George Cybenko. “Approximation by Superpositions of a Sigmoidal Function”. In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), pp. 303–314.

[Dhi$^+$16] Bhuwan Dhingra et al. “Towards End-to-End Reinforcement Learning of Dialogue Agents for Information Access”. In: (2016). arXiv: `1609.00777`.

[DL16] Li Dong and Mirella Lapata. “Language to Logical Form with Neural Attention”. In: (2016). arXiv: `1601.01280`.

[Dod$^+$15] Jesse Dodge et al. “Evaluating Prerequisite Qualities for Learning End-to-End Dialog Systems”. In: (2015), pp. 1–16. arXiv: `1511.06931`.

[EM17] Mihail Eric and Christopher D. Manning. “Key-Value Retrieval Networks for Task-Oriented Dialogue”. In: (2017). arXiv: `1705.05414`.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016, p. 800. ISBN: 9780262035613. URL: `http://www.deeplearningbook.org`.

[GJM13] Alex Graves, Navdeep Jaitly, and Abdel Rahman Mohamed. “Hybrid speech recognition with Deep Bidirectional LSTM”. In: *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2013 - Proceedings* (2013), pp. 273–278.

[GMH13] Alex Graves, Abdel Rahman Mohamed, and Geoffrey E. Hinton. “Speech Recognition with Deep Recurrent Neural Networks”. In: 3 (2013). arXiv: `1303.5778`.

[Gre$^+$17] Klaus Greff et al. “LSTM : A Search Space Odyssey”. In: (2017), pp. 1–12. arXiv: `1503.04069v2`.

[GRW97] A. Gorin, G. Riccardi, and J. Wright. “How may I help you?” In: *Speech Communication* 23.1-2 (1997), pp. 113–127.

[GS00] Felix A. Gers and Jürgen Schmidhuber. “Recurrent nets that time and count”. In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* February 2000 (2000), 189–194 vol.3. arXiv: `1011.1669v3`.

[GS05] Alex Graves and Jürgen Schmidhuber. “Framewise Phoneme Classification with Bidirectional {LSTM} Networks”. In: *Proc. Int. Joint Conf. on Neural Networks IJCNN 2005* (2005).

[GSC99] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to Forget: Continual Prediction with LSTM”. In: (1999), pp. 850–855.

[HAK16] Homa B. Hashemi, Amir Asiaee, and Reiner Kraft. “Query Intent Detection using Convolutional Neural Networks”. In: (2016). DOI: `10.1145/1235`.

[Hoc91] Sepp Hochreiter. “Untersuchungen zu dynamischen neuronalen Netzen”. PhD thesis. 1991, pp. 1–71.

[Hoc$^+$01] Sepp Hochreiter et al. “Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies”. In: *A Field Guide to Dynamical Recurrent Networks. IEEE Press* (2001). arXiv: `1011.1669v3`.

[Hon16] Matthew Honnibal. *Embed, encode, attend, predict : The new deep learning formula for state-of-the-art NLP methods*. 2016. URL: `https://explosion.ai/blog/deep-learning-formula-nlp` (visited on 05/24/2018).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural Computation* 9.8 (1997), pp. 1–32. arXiv: `1206.2944`.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. arXiv: `1011.1669v3`.

[HTY13] Matthew Henderson, Blaise Thomson, and Steve Young. “Deep Neural Network Approach for the Dialog State Tracking Challenge”. In: *Proceedings of the SIGDIAL 2013 Conference* (2013), pp. 467–471. URL: `http://www.aclweb.org/anthology/W/W13/W13-4073`.

[Irp18] Alex Irpan. *Deep Reinforcement Learning Doesn’t Work Yet*. 2018. URL: `https://www.alexirpan.com/2018/02/14/rl-hard.html` (visited on 06/24/2018).

[JM17] Daniel Jurafsky and James H Martin. “Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition”. In: (2017). URL: `https://web.stanford.edu/{~}jurafsky/slp3/ed3book.pdf`.

[KLM96] Leslie Kaelbling, Michael L. Littman, and Andrew W. Moore. "Reinforcement Learning: A Survey". In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 1475 –1479.

[Kou+13] Jan Koutník et al. "Evolving large-scale neural networks for vision-based reinforcement learning". In: *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13* (2013), p. 1061.

[LBH15] Yann Lecun, Yoshua Bengio, and Geoffrey E. Hinton. "Deep learning". In: (2015).

[Lea10] Neal Leavitt. "Will NoSQL Databases Live Up to Their Promise?" In: *IEEE Computer Society* 43.2 (2010), pp. 12–14.

[Li+17] Xiujun Li et al. "End-to-End Task-Completion Neural Dialogue Systems". In: (2017). arXiv: `1703.01008`.

[Lia16] Percy Liang. "Learning Executable Semantic Parsers for Natural Language Understanding". In: (2016). arXiv: `1603.06677`.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: (2015). arXiv: `1508.04025`.

[LR85] T. L. Lai and Herbert Robbins. "Asymptotically efficient adaptive allocation rules". In: *Advances in Applied Mathematics* 6.1 (1985), pp. 4–22. DOI: `10.1016/0196-8858(85)90002-8`.

[MC68] D. Michie and R. A. Chambers. "BOXES: An Experiment in Adaptive Control". In: *Machine Intelligence 2* (1968), pp. 137–152.

[Mik12] Tomas Mikolov. "Statistical Language Models Based on Neural Networks". PhD thesis. 2012, pp. 1–129. ISBN: 0885-2308. arXiv: `1312.3005`.

[Mik+13a] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: (2013), pp. 1–9. arXiv: `1310.4546`.

[Mik+13b] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *Proceedings of the International Conference on Learning Representations (ICLR 2013)* (2013), pp. 1–12. arXiv: `1301.3781v3`.

[Mil+16] Alexander H. Miller et al. "Key-Value Memory Networks for Directly Reading Documents". In: (2016). arXiv: `1606.03126v2`.

[Mil+17] Alexander H. Miller et al. "ParlAI: A Dialog Research Software Platform". In: (2017). arXiv: `1705.06476`.

[Mit97] Tom Mitchell. *Machine learning.* 1997. ISBN: 0070428077.

[Mni$^+$13] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: (2013), pp. 1–9. ISSN: 0028-0836. arXiv: `1312.5602`.

[Mni$^+$15] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533. URL: `http://dx.doi.org/10.1038/nature14236`.

[MP43] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. arXiv: `1011.1669v3`.

[Mrk$^+$16] Nikola Mrksic et al. “Neural Belief Tracker: Data-Driven Dialogue State Tracking”. In: (2016). arXiv: `1606.03777`.

[Nie15] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[NPP13] Ameya Nayak, Anil Poriya, and Dikshay Poojary. “Type of NOSQL Databases and its Comparison with Relational Databases”. In: *International Journal of Applied Information Systems* 5.4 (2013), pp. 16–19.

[Ola15] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/` (visited on 05/23/2018).

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training Recurrent Neural Networks”. In: (2013). arXiv: `1211.5063`.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “Glove: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1532–1543. arXiv: `1504.06654`.

[Raj$^+$16] Pranav Rajpurkar et al. “SQuAD: 100,000+ Questions for Machine Comprehension of Text”. In: ii (2016). arXiv: `1606.05250`.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. arXiv: `1011.1669v3`.

[RN03] J. Russell, Stuart and Peter Norvig. *Artificial Intelligence a Modern Approach*. Vol. 72. 1-2. 2003, pp. 81–138. ISBN: 0137903952.

[Ros58] F Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain”. In: *Psychological Review* 65.6 (1958), pp. 386–408. arXiv: `1112.6209`.

[Rud16] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: (2016), pp. 1–14. arXiv: `1609.04747`.

[Sal⁺18] Hojjat Salehinejad et al. “Recent Advances in Recurrent Neural Networks”. In: (2018), pp. 1–21. arXiv: `1801.01078`.

[SB17] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction -*. Vol. 3. 9. 2017, p. 360. ISBN: 0262193981. arXiv: `1603.02199`.

[Ser⁺15] Iulian Vlad Serban et al. “A Survey of Available Corpora for Building Data-Driven Dialogue Systems”. In: (2015). arXiv: `1512.05742`.

[Sil15] David Silver. *Lecture 7 : Policy Gradient*. 2015. URL: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching{\_}files/pg.pdf` (visited on 05/23/2018).

[Sil⁺16] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489. arXiv: `1610.00633`.

[Ska07] Gabriel Skantze. “Error Handling in Spoken Dialogue Systems”. PhD thesis. 2007, p. 197. ISBN: 9781598295993. DOI: `10.1007/978-3-540-49127-9_35`.

[Sol18] SolidIT. *DB-Engines Ranking*. 2018. URL: `https://db-engines.com/en/ranking` (visited on 05/24/2018).

[Sri⁺14] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. arXiv: `1102.4807`.

[Suk⁺15] Sainbayar Sukhbaatar et al. “End-To-End Memory Networks”. In: (2015), pp. 1–11. arXiv: `1503.08895`.

[Sut⁺99] Richard S. Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *In Advances in Neural Information Processing Systems 12* (1999), pp. 1057–1063.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to sequence learning with neural networks”. In: *Advances in Neural Information Processing Systems (NIPS)* (2014), pp. 3104–3112. arXiv: `1409.3215`.

[Tes92] Gerald Tesauro. “Practical Issues in Temporal Difference Learning”. In: *Machine Learning* 8.3 (1992), pp. 257–277.

[Tra16] Tractica. *The Virtual Digital Assistant Market Will Reach $15.8 Billion Worldwide by 2021*. 2016. URL: `https://www.tractica.com/newsroom/press-releases/the-virtual-digital-assistant-market-will-reach-15-8-billion-worldwide-by-2021/` (visited on 05/22/2018).

[VFJ15] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. “Pointer Networks”. In: (2015), pp. 1–9. arXiv: `1506.03134`.

[WCB14] Jason Weston, Sumit Chopra, and Antoine Bordes. “Memory Networks”. In: (2014), pp. 1–15. arXiv: `1410.3916`.

[Wei66] Joseph Weizenbaum. “ELIZA — A Computer Program For the Study of Natural Language Communication Between Man And Machine”. In: *Communications of the ACM* 9.1 (1966), pp. 36–45.

[Wen$^+$16] Tsung-Hsien Wen et al. “A Network-based End-to-End Trainable Task-oriented Dialogue System”. In: (2016). arXiv: `1604.04562`.

[Wer90] J. Paul Werbos. “Backpropagation through time: what it does and how it does it”. In: (1990).

[Wes$^+$15] Jason Weston et al. “Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks”. In: *Journal of Power Sources* 273 (2015), pp. 486–494. arXiv: `1502.05698`.

[Wil92] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Machine Learning* 9 (1992), pp. 229–256.

[WKNw72] William Woods, R. M. Kaplan, and B. L. Nash-webber. “The Lunar Science Natural Language Information System: Final Report”. In: *ResearchGate* January 2016 (1972).

[WZ16] Jason D. Williams and Geoffrey Zweig. “End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning”. In: (2016). arXiv: `1606.01269`.

[XLS17] Xiaojun Xu, Chang Liu, and Dawn Song. “SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning”. In: (2017), pp. 1–13. arXiv: `1711.04436`.

[Xu$^+$15] Kelvin Xu et al. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: (2015). arXiv: `1502.03044`.

[Yag17] Navid Yaghmazadeh. “SQLizer: Query Synthesis from Natural Language”. In: *Proc. ACM Program. Lang* 1.26 (2017).

[Yan$^+$17] Zhao Yan et al. “Building Task-Oriented Dialogue Systems for Online Shopping”. In: *Proceedings of the 31th Conference on Artificial Intelligence (AAAI 2017)* (2017), pp. 4618–4625.

[YD15] Dong Yu and Li Deng. *Automatic Speech Recognition.* Signals and Communication Technology. London: Springer London, 2015, p. 321.

[YYM15] Yi Yang, Wen-Tau Yih, and Christopher Meek. “WIKIQA: A Challenge Dataset for Open-Domain Question Answering”. In: *Proceedings of EMNLP 2015* September 2015 (2015), pp. 2013–2018.

[ZF14] Matthew D. Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8689 LNCS.PART 1 (2014), pp. 818–833. arXiv: `1311.2901`.

[ZXS17] Victor Zhong, Caiming Xiong, and Richard Socher. “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning”. In: (2017), pp. 1–12. arXiv: `1709.00103`.

Assertion

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Ebenso versichere ich, dass diese Arbeit oder Teile daraus weder von mir selbst noch von anderen als Leistungsnachweise andernorts eingericht wurde. Sämtliche Sekundärliteratur und sonstige Quellen sind nachgewiesen und in der Bibliographie aufgeführt. Das Gleich gilt für graphische Darstellungen und Bilder sowie für alle Internet-Quellen. Mir ist bekannt, dass von der Korrektur der Arbeit abgesehen werden kann, wenn die Erklärung nicht erteilt wird.

Karlsruhe, June 27, 2018 Blank Sebastian