The Reasoning and Learning Lab Chatbot:

a solution to the Conversational Intelligence Challenge

Nicolas Angelard-Gontier

260532513

School of Computer Science
McGill University, Montreal

August 1, 2018

Acknowledgements

I would like to thank everyone with whom I had the chance to share my enthusiasm upon exploring new concepts; those who gave me the opportunity to learn, think and explore new ideas.

In particular, I would like to express my deepest gratitude to my supervisor Professor Joelle Pineau for her precious insights, her unwavering support and mentorship throughout this thesis. In addition, I want to thank all the other great professors from whom I had the opportunity to learn a lot, in alphabetical order: Doina Precup, Jackie Cheung, Luc Devroye, Simon Lacoste-Julien, Timothy O'Donnell. I must also acknowledge great help from my friends and colleagues, in alphabetical order: Iulian Serban, Koustuv Sinha, Michael Noseworthy, Nan Rosemary Ke, Peter Henderson, Prasanna Parthasarathi, Ryan Lowe.

Finally, the work reported in this thesis would not have been possible without the financial support from: NSERC, Calcul Quebec, Compute Canada and the Facebook AI Research ParlAI fund.

Abstract

Over the years, the richness of human computer interaction has greatly evolved. As a result, it is now possible to command a computer, phone, or watch using natural language. Recently, the idea of having a text-based conversation with a machine became more feasible. This is in particular due to modern developments in Deep Learning, a quickly growing area of research in the field of Artificial Intelligence that has achieved impressive results in various applications such as conversational intelligence.

Current dialog systems are able to understand our queries and do simple tasks such as listing the news, obtaining the weather or playing music. A significant amount of efforts have been made to train such systems. A logical next step is to be able to have a relatively long and entertaining conversation with chatbots in order to solve more complex tasks. Competitions like the *Conversational Intelligence* (*ConvAI*) challenge[1] are being organized to push the research development towards that goal.

This thesis undertakes an analysis of modern dialog systems that use deep learning tools. More precisely, we present in details the *Reasoning and Learning Lab Chatbot* (*RLLChatBot*) that participated in the *ConvAI* challenge. We analyze different text generation models and the important role of a ranking system in order to regulate the conversation flow. The main contribution of this work is that we provide a thorough description of how a dialog system can be built and trained from mostly public-domain datasets using Deep Learning tools. Moreover, we present an additional dataset we collected, allowing to improve our ranking system that evaluates which candidate response should be returned to the user at each time step in the conversation. We leave as future work the goal of unifying the entire pipeline we present, as of now, each model is trained independently.

[1] http://convai.io/2017/

Résumé

Au fil des ans, la richesse de l'interaction homme-machine a beaucoup évolué. Par conséquent, il est maintenant possible de commander un ordinateur, un téléphone ou une montre en utilisant le langage naturel. Récemment, l'idée d'avoir une conversation avec une machine est devenue réalisable. Ceci est dû en particulier aux développements modernes d'apprentissage profond, une dicipline de recherche en croissance rapide dans le domaine de l'intelligence artificielle qui a obtenu des résultats impressionnants dans diverses applications telles que l'intelligence conversationnelle.

Les systèmes de dialogue actuels sont capables de comprendre nos requêtes et de faire des tâches simples telles que lister les nouvelles, obtenir la météo ou jouer de la musique. Des efforts importants ont été déployés pour former de tels systèmes. Une prochaine étape logique est de pouvoir avoir une conversation relativement longue et divertissante avec les chatbots afin de résoudre des tâches plus complexes. Des compétitions comme le *Conversational Intelligence* (*ConvAI*) challenge[2] sont organisées pour pousser le développement de la recherche vers cet objectif.

Cette thèse entreprend une analyse des systèmes de dialogue qui utilisent l'apprentissage profond. Plus précisément, nous présentons le *Reasoning and Learning Lab Chatbot* (*RLLChatBot*) qui a participé au défi *ConvAI*. Nous analysons différents modèles de génération de texte et le rôle important d'un système de classement afin de réguler la conversation. La principale contribution est un système de dialogue entrainé à partir d'ensembles de données publiques en utilisant des outils d'apprentissage profond. De plus, nous présentons un nouvel ensemble de données permettant d'améliorer notre système de classement qui évalue quelle réponse doit être retournée à l'utilisateur. Nous laissons comme futur travail l'objectif d'unifier l'ensemble des modèles que nous présentons, pour l'instant, chaque modèle est entrainé indépendamment.

[2] http://convai.io/2017/

Contents

Contents **iv**

List of Figures **vii**

1 Introduction **1**

1.1 Overview of Dialog Systems 2

1.1.1 Goal Oriented Dialog Systems 2

1.1.2 Open Domain Dialog Systems 3

1.2 Conversational Challenge Description 5

1.3 Summary of Contributions 8

2 Previous Work in the Chatbot Community **10**

2.1 History of Dialog Systems 10

2.2 History of ChatBot Competitions 14

3 Technical Preliminaries **17**

3.1 Artificial Neural Networks 17

3.1.1 Artificial Neuron 18

3.1.2 Activation Function 19

3.1.3 Multi-Layer Perceptron 21

3.2 Recurrent Neural Networks 22

3.2.1 Recurrent Neural Networks . . . 23
3.2.2 Long Short-Term Memory Networks . . . 25
3.2.3 Gated Recurrent Unit Networks . . . 27
3.3 Training Artificial Neural Networks . . . 28
3.3.1 Cost Functions . . . 28
3.3.2 Gradient Based Optimization . . . 30
3.3.3 Backpropagation Algorithm . . . 32
3.4 Reinforcement Learning Introduction . . . 33
3.4.1 Markov Decision Process . . . 35
3.4.2 Q-learning and Fitted Q-Iteration . . . 35

4 Data-Driven Dialog Systems 39
4.1 Natural Language Interpreter . . . 40
4.1.1 Utterance Classification . . . 41
4.1.2 Sequence Tagging . . . 41
4.2 Dialog State Tracker . . . 42
4.3 Dialog Manager . . . 44
4.4 Natural Language Generator . . . 45

5 Generation of Candidate Responses 47
5.1 Fully Generative systems . . . 48
5.1.1 Hierarchical Recurrent Encoder Decoder . . . 49
5.1.2 Neural Question Generator . . . 50
5.2 Retrieval-based systems . . . 53
5.2.1 DrQA . . . 53
5.2.2 Topic Classifier . . . 54
5.2.3 Fact Retriever . . . 56
5.3 Rule-based systems . . . 57
5.3.1 Entity Sentences . . . 58

5.3.2 Simple Answers 59
5.3.3 A.L.I.C.E. Bot 60

6 Scoring and Selection of Responses 61
6.1 Scoring of candidate responses 61
6.1.1 Supervised Scoring 62
6.1.2 Q-Scoring 65
6.2 Selecting one response 68

7 Data Extension and Experiments 75
7.1 Data Collection 76
7.2 Experimentation and Evaluation 82

8 Conclusion 95
8.1 Summary 95
8.2 Limitations 97
8.3 Future Work 98

Bibliography 100

List of Figures

1.1 Example of bot-to-human conversation. 7

3.1 A graphical illustration of an artificial neuron. 18
3.2 Different activation functions. 20
3.3 A three layer fully connected, feed-forward neural network. 21
3.4 A computation graph of an RNN. 23
3.5 A graphical description of the gradient descent algorithm. 30
3.6 Forward and backward pass in the backpropagation algorithm. 32
3.7 Typical reinforcement learning loop between an agent and its environment. 34

4.1 Pipeline framework of modular dialog systems. 40

5.1 High-level view of our chatbot. 48
5.2 The Hierarchical Recurrent Encoder Decoder model. 49

6.1 Feed-forward network to predict candidate responses' vote. 62
6.2 Proportion of end-of-conversation score given by human users. 66
6.3 Deep Q-Network to predict candidate responses' action value. 68
6.4 Rules followed during a conversation in order to decide which response to return to the user. 74

7.1 User interface of *Amazon Mechanical Turk* during our data collection. . . 77

7.2 Statistics on the data collected with *Amazon Mechanical Turk*. 78

7.3 Statistics on the different candidate model usage. 79

7.4 Recall measurements for different selection mechanism with our baseline model. 87

7.5 Recall measurements for different selection mechanisms with the best *SmallR* and *DeepR* models. 89

7.6 Recall measurements for different selection mechanism with the best *SmallQ* and *DeepQ* models. 91

1

Introduction

> – Les lois de la conversation sont en général de ne s'y appesantir sur aucun objet, mais de passer légèrement, sans effort et sans affectation, d'un sujet à un autre ; de savoir y parler de choses frivoles comme de choses sérieuses
> – The rules of conversation are, in general, not to dwell on any one subject, but to pass lightly from one to another without effort and without affectation ; to know how to speak about trivial topics as well as serious ones
>
> *The Encyclopedia of Diderot, Jean le Rond d'Alembert, start of the entry on 'Conversation', in Grammar category, in Logic parent-category; 1752*

Having a conversation with computers is not a novel idea. Already in the 1950's Allan Turing was thinking about it (Turing, 1950). Not long after, Weizenbaum built the first computer program that could interact with humans using natural language (Weizenbaum, 1966). A couple of years later, computation with layered networks of artificial neurons were developed (Levin and Fleisher, 1988; Hornik, Stinchcombe, and H. White, 1989). However such networks require a lot of data to be trained and thus they were not used for dialog systems until recently. Recent progress in computer hardware and the availability of big amounts of data changed our approach to building dialog systems. We now have powerful enough computers and sufficient amounts of public conversations to build data-driven conversational systems. This thesis demon-

strates the potential of deep data-driven architectures to maintain conversations with humans.

In this chapter we define what dialog systems are, describe the conversational challenge, and outline the different contributions of this thesis.

1.1 Overview of Dialog Systems

We define a dialog system to be a computer program that is responsible for returning an output sentence to one or more input sentences. We also denote them as *Conversational Agents*, or *Chatbots* (Weizenbaum, 1966; Colby, 1975; Epstein, 1993; Serban, Sankar, et al., 2017). They communicate via Natural Language by using speech and/or text signals. In this work we focus on text-to-text interactions only. This simplifies the task a little as going from a text-based chatbot to a spoken system can be challenging due to speech recognition errors. Dialog systems can be in multi-agent settings where each agent is either a human or another system. Here we restrain again the setting by only allowing one human and one virtual agent in the environment. We thus focus on bot-to-human interactions only.

In general, we cluster conversational agents into two distinct categories: *goal-oriented* systems and *open-domain* systems. We now define each setting to better understand the landscape of dialog systems research.

1.1.1 Goal Oriented Dialog Systems

In the goal-oriented setting, the system is explicitly built to solve a task (McGlashan et al., 1992; Aust et al., 1995; Gorin, Riccardi, and Wright, 1997). Typical conversations follow roughly the same structure: the user queries the chatbot; assuming the chatbot understood, it will ask clarifying questions before giving a final answer and confirm if an action has been taken. Such conversations are usually quite short and, to the extreme, can be one interaction long. These systems include all the digital assis-

tants we have in modern cell-phones, computers and house controllers (Alexa, Google Assistant, Siri, Cortana, etc.). They can solve simple and specific tasks for the user such as setting an alarm clock, calling someone, playing music, giving the weather, finding restaurants, or buying various common items in their associated on-line store.

In general, goal-oriented agents can be thought of as an alternate user interface for any application. Instead of using your hands to command the machine, you can now use your voice. These systems play an important role in particular for customer service applications where companies want to help customers answer questions or solve problems. Goal-oriented dialog systems operate in well-defined domains, they are structured and logical. Using rule-based systems can achieve good results for the task that they are programmed to do, but nothing more. One major weakness of goal-oriented systems is that they are good for one or a few specific tasks but as soon as you switch domain, the chatbot is not of much help (Simpson and Eraser, 1993). While solving just a few tasks at a time may seem reasonable for some applications, this does not help us understand how humans learn to speak. In addition, with enough rules, conversational agents are able to correctly answer a wide variety of responses, however in a realistic example, there is never enough rules to cover all possible conversational scenarios that can be imagined by a human. Therefore, while being accurate, goal-oriented chatbots often lack personalization and flexibility.

1.1.2 Open Domain Dialog Systems

In the open-domain setting, the system is not meant to solve specific tasks, rather its role is to be a social companion to the user (Weizenbaum, 1966; Colby, 1975; Epstein, 1993; Hutchens and Alder, 1998). Rather than achieving some goal, the chatbot has to mimic as much as possible the unstructured characteristic of human-to-human conversations, while still being coherent. We want the robot to '*sound human*', in other words, we want the robot to pass the Turing test as originally described

by Turing (1950). Alan Turing wanted to answer this fundamental question: "*Can machines think?*". Applied to chatbots, the test looks like this: imagine a scenario with two separate rooms, you are in one room and you try to guess if there is a human or a computer in the other room. To make your decision you chat with the unknown entity in the other room. If you think that you are talking to a human but in fact it was a computer generating answers, then the computer successfully passed the Turing test (also called the "*imitation game*").

Unlike in the goal-oriented setting, typical conversations in the open domain setting are longer than just a few interactions. Open domain dialog systems are mainly used for entertainment purposes such as role-playing (virtual characters in games and social platforms; for instance Microsoft's Xiaoice, Tay, and Zo) but they can also be used for more practical reasons such as testing psychological theories (Weizenbaum, 1966; Colby, 1975). In general open domain dialog agents aim at being more '*social*' than goal oriented systems in order to facilitate human-to-computer interactions and make machine interfaces more user-friendly. Because of their unstructured setting, open domain agents are much more flexible and can be used to better understand how humans discuss. Indeed, without any achievement in mind it is harder to simply use logical rules to guide the generation of responses; we have to focus on something else, we have to understand how humans speak.

However, it is worth mentioning that the lack of a clear task to achieve makes it hard to automatically evaluate conversational agents in this setting. Unlike in the goal-oriented case, here we do not have to achieve an objective, thus it is harder to check at the end of a conversation if it was successful or not. This remains an open problem in the field and to this day the best evaluation for open domain chatbots is to ask humans to manually score conversations (C.-W. Liu et al., 2016). This is one major limitation of open domain dialog agents, and this is why it is important to organize competitions in this field.

1.2 Conversational Challenge Description

Challenges are a great way to push the frontiers of science (J. Williams, Raux, and Henderson, 2016). In a fast growing field such as deep learning, competitions are popular because they identify what current research can do, which technique work or does not work, and the next challenges to be solved by the community. For instance, the *Pascal CHiME* challenge in speech recognition (Barker et al., 2013), the *ImageNet Large Scale Visual Recognition* challenge (Russakovsky et al., 2015) and the *Stanford Question Answering Dataset* in machine comprehension (Rajpurkar et al., 2016) are all examples of popular competitions that allowed the development of new algorithms and models to beat the state of the art in their respective fields. In addition, competitions in artificial intelligence can also tell what humans can do better than machines. Often times, the human accuracy is the gold standard level to achieve or surpass. Having an objective to accomplish is often a source of motivation for researchers. As mentioned previously, competitions in the field of conversational AI date back to 1950s with the Turing Test (Turing, 1950). Since then, many competitions reframed the problem in various ways. Most notably, Cambridge Centre for Behavioural Studies proposed a first formal instantiation of the Turing Test in 1991. The test is known as the *Loebner Prize* as Dr. Hugh Loebner pledged a $100,000 grand prize for the first computer program to pass the test (Epstein, 1993; Hutchens and Alder, 1998). More recently, the *Amazon Alexa Prize*[1] challenged universities to discuss in real time with some Alexa users. This competition's goal is to have the longest conversation possible while being coherent (Serban, Sankar, et al., 2017). The *Conversational Intelligence* (*ConvAI*) challenge is a more defined, text-based version of the *Alexa Prize*: the topic of the discussion is defined at the beginning of each dialog and every conversation is on

[1] https://developer.amazon.com/alexaprize

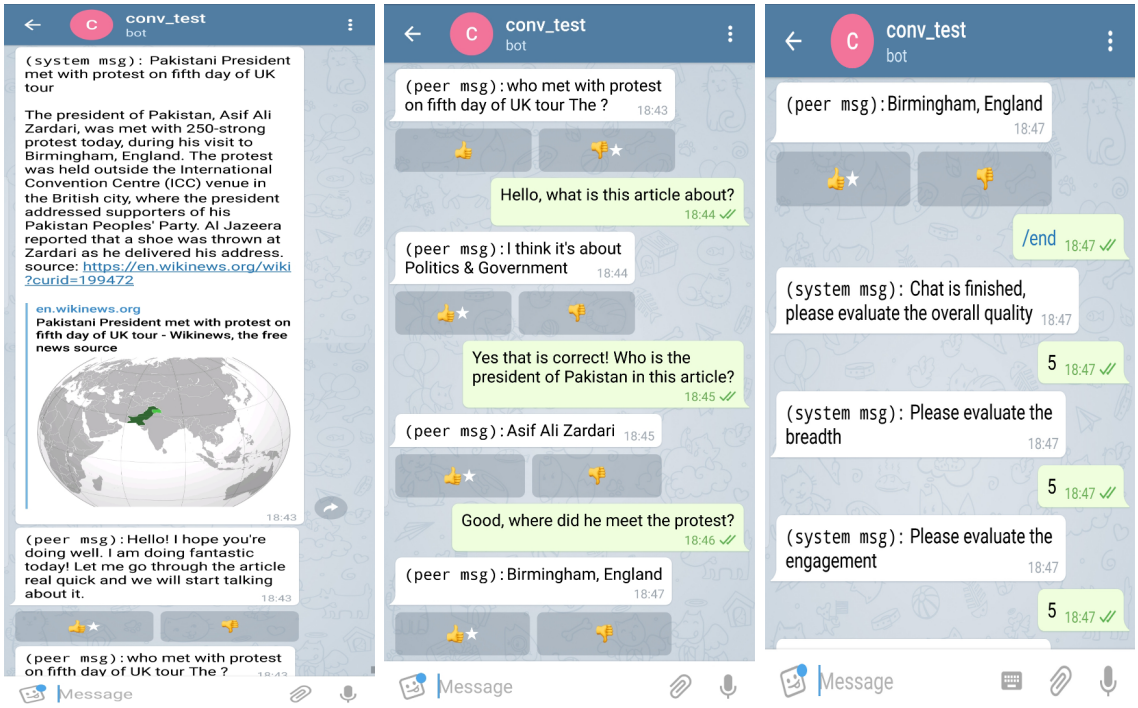
text messaging platforms such as Telegram[2]. Overall, participating in these challenges is important because it motivates researchers to surpass the current state of the art and gives human evaluations to submitted systems, which can then be used to improve initial solutions.

The proposed work is a description of our dialog system build for the *Conversational Intelligence* (*ConvAI*) challenge organized as part of the competition workshop of the 2017 *Neural Information Processing Systems* (*NIPS*) conference. The conference is the largest conference in Artificial Intelligence and Machine Learning in terms of registered participants. Besides machine learning, other fields represented at *NIPS* include cognitive science, psychology, computer vision, statistical linguistics, and information theory. In December 2017, the conference had a competition workshop track in which many challenges took place. The *ConvAI* challenge was one of them.

As participants to the challenge, we had to build a conversational system that can talk to human judges about a random wiki-news article's paragraph. Like the Turing test described above, at the beginning of each conversation, human judges do not know if they are talking to a bot or another human. After each message, human users can 'up-vote' or 'down-vote' individual responses of the other participant. The two participants chat for any number of interactions desired, keeping in mind the news paragraph given at the very beginning of the conversation. At the end of the conversation human users give a score between 1 and 5 for the conversation quality, breadth, and engagement (1 being 'very bad', 2 'bad', 3 'medium', 4 'good', 5 'very good'). Submitted systems are then given the average score among all the conversations they had with random human users. After many rounds of evaluation, the organizers collected a dataset of human-to-human and human-to-bot conversations, each evaluated from 1 to 5. Figure 1.1 is an example of how a conversation looked like during evaluation:

We recognize this scenario as an instance of the previously described open domain

[2] https://telegram.org/



(a) begining of conversation: the first message is a random article to be discussed by the two participants.

(b) middle of conversation with messages being voted by the human participant.

(c) end of conversation with evaluation of the conversation by the human participant.

Figure 1.1: Example of bot-to-human conversation during the *ConvAI* evaluation using the *Telegram* platform.

setting. Indeed, the task here is to simply chat about any given news paragraph with a human. The organizers restrained the domain a little in each conversation by giving a random article to talk about, but beside this, no extra information was given. It is important to note that each new conversation had a unique random news paragraph with topics ranging from sports, politics, science, history, technology, fashion, economics, and many others. The submitted system has to be general enough to understand and speak about all these topics. Moreover, the challenge rules explicitly stated that we are not allowed to connect our chatbot to the internet. We could not query external knowledge bases on the web for each new article or for each message the system received. Therefore, the difficulty of the task is to extract information from the article as soon as we receive it, and to be able to have a coherent conversation

about it without any other external information.

The work in this thesis describes all the components of the chatbot submitted for this competition as well as extra modules that were added after the competition. The *ConvAI* challenge was a good motivation for us to build a working system. Part of the challenge was the engineering effort of exporting a research project into a real system with all the constraints that comes with it such as latency, concurrency, memory, and others. The primary goal of this research was to better understand how current deep learning and reinforcement learning tools can be used to make a robust yet flexible open domain chatbot.

1.3 Summary of Contributions

This thesis describes all the components of the *RLLChatBot* submitted for the *ConvAI* competition as well as extra components that were added after the competition. We also introduce a new conversational dataset that we collected after the challenge to train those extra components. The goal of this research is to better understand how current deep learning and reinforcement learning tools can be used to make a robust yet flexible open domain chatbot. Participating in this competition allowed us to thoroughly assess and benchmark the system, in a way that enhances our understanding of chatbots.

We start in Chapter 2 with a literature review in which we cover the previous work done in the field of conversational systems. In Chapter 3 we introduce fundamentals of artificial neural networks, the gradient-based way of training them and their application in dialog systems. In particular, we present recurrent neural networks as they are the architecture of choice when working on sequential data such as text. In Chapter 4 we give a detailed description of a typical modular architecture for data-driven dialog systems. We explain some limitations and advantages of such architecture. Chapters 5 provides a thorough description of our chatbot system by

decomposing it into different modules. We first focus on a variety of generative-, retrieval-, and rule-based models to produce a set of candidate responses. We then explore scoring and selection strategies in Chapter 6. More specifically, we introduce one supervised and one reinforcement learning strategy for our scoring mechanism, and a rule-based selection criteria used during the competition. Throughout our own data collection and experiments described in Chapter 7, we demonstrate that the ranking mechanism plays a crucial role as it decides which response to return at each step in a conversation.

The first contribution of this work is the full conceptualization of a data-driven dialog system using a combination of rule based models, and trainable retrieval and generative models. None of these models is allowed to query the internet to get external information. The second contribution is the novel conversational dataset we collected after the challenge, and most importantly, the different scoring mechanisms trained on this data that select which response to return to the user at each time step in a conversation. Both greedy and sophisticated approaches are explored.

2

Previous Work in the Chatbot Community

> – Success depends upon previous preparation, and without such preparation there is sure to be failure.
>
> *Confusious*

After introducing goal-oriented and open-domain dialog systems as well as the *ConvAI* competition, we review previous work that has been done in the chatbot community. We start by providing an overview of previously build chatbots. We then introduce various competitions that pushed the field of research forward, and tried to evaluate conversational agents.

2.1 History of Dialog Systems

The original idea of naturally and meaningfully communicating with a machine began to flourish in the 1950s when Alan Turing proposed the *Turing Test* (Turing, 1950). Having systems that can communicate with humans has been a central goal of artificial intelligence ever since.

The very first chatbot was built in 1966 by MIT Scientist Joseph Weizenbaum, and was named *ELIZA* (Weizenbaum, 1966). Entirely rule-based, the *ELIZA* program analyze its input sentences with decomposition rules that are triggered by keywords. Responses are simply sampled from a list of matching scripted sentences. This first

simple approach to building conversational agents yields conversations that feel like talking to a Rogerian psychotherapist that performs a person-centered therapy, where the system is mostly repeating something the user said or asking to talk more about it.

Shortly after this first chatbot, Standford psychiatrist Kenneth Colby developed another system able to communicate via natural language text: *PARRY* (Colby, 1975). Also rule-based, this conversational agent is built to simulate as accurately as possible the thinking patterns of a paranoid schizophrenic. Often described as "ELIZA with attitude", this chatbot produces more serious conversations.

Following the "AI winter" in the 1980s during which progress and research slowed down in the field of Artificial Intelligence, the 1988 *Jabberwacky* chatbot was built. It is one of the earliest attempts at creating a conversational system from human conversations. Very different from its predecessors, this system is not entirely based on conversation rules, rather, it stores everything that everyone has ever said to it and finds the most appropriate thing to reply based on contextual pattern matching techniques (Fryer and Carpenter, 2006). This allows the chatbot to become more flexible and even chat in other languages if enough conversations are in its database. After being published on the web in 1997, its database increased to more than 10 million conversations in a few years[1]. This system can be seen as the first data-driven chatbot.

Not long after that, one of the most famous chatbot was made by computer scientist Dr. Richard Wallace: the *Artificial Linguistic Internet Computer Entity*, also known as *ALICE*. Inspired by *ELIZA*, this 1995 chatbot is one of the strongest rule-based agent with more than 20,000 conversational rules. This is the first program to rely on the XML schema called AIML (Artificial Intelligence Markup Language) for specifying heuristic conversation rules (Wallace, 2009). AIML contains several elements such as *categories*, *patterns* and *templates*. Categories form the fundamental

[1] http://www.jabberwacky.com/

unit of knowledge, they can be considered as one conversational rule in the case of ALICE. Each category consists of at least one pattern and one template element, where the pattern represents a string of characters intended to match one or more user inputs, and the template represents the response to a matched pattern. An important thing to note is that AIML handles recursion (in which a template can refer to another pattern) and variable objects that store dynamic information like user names. All this makes ALICE a strong and flexible chatbot and allowed it to win the Loebner Prize three times in 2000, 2001, and 2004.

The very first chatbot to become a widely used consumer product was *SmarterChild*, developed by *ActiveBuddy*, a company that created conversation-based interactive agents originally distributed via instant messaging platforms. In 2001, the dialog agent was released on AOL Instant Messenger and Windows Live Messenger networks as a showcase for the quick data access and possibilities for fun personalized conversation (Kay and Hoffer, 2006; Kay and Hoffer, 2007; Cunningham, Pache, and C. White, 2007). Being a true social bot, *SmarterChild* attracted over 30 million users. In addition to relying on the same concept of pattern matching and scripted answers that *ELIZA* used decades earlier, the innovative aspect of this bot is that it can provide useful information via partnership with various service providers to offer weather, stocks, movie listings and more.

During the early 2000s, chatbots started to rely less on hand-crafted rules and more on data-driven approaches (Lester, Branting, and Mott, 2004). This shift was primarily caused by the growing abundance of conversational data with the introduction of new communication technologies via the Internet. In general, the conversational agents that are developed during this time are following a pipeline (or modular) architecture as described by Rudnicky et al. (1999), S. J. Young (2000), and Zue and Glass (2000) in which user queries are first parsed and interpreted, then a dialog manager have the role of providing response elements, before a response generator module can return a proper sentence. We describe this pipeline in more details in Chapter 4.

In 2006, a team of IBM researchers started to build a computer system to compete in the *Jeopardy* challenge: *IBM's Watson*. The goal of the challenge is to answer a wide variety of questions as fast as possible on various topics. In 2010, the system was ready to compete and won the challenge over two previous champions. The system relies on the *DeepQA* architecture, combining many search algorithms that analyze the data under many different perspectives and consider all possible evidence with respect to hundreds of different answers (Ferrucci et al., 2010). The output of the *DeepQA* architecture contains a confidence score along with a response, and depending on the confidence, *Watson* replies or not. Even-though this technology advanced the field of Question Answering specifically, it is a notable milestone in the broader field of conversational artificial intelligence as question answering can be framed as a sub-task of the dialog task.

At the same time, a start-up called *Siri* was getting acquired by tech giant *Apple* and became the most popular chatbot to this day. Being a voice-activated system, it uses innovative speech recognition techniques to transform speech to text before analyzing user queries and providing appropriate responses in a pipeline manner (similar to techniques presented by Rudnicky et al. (1999) and described in Chapter 4). This new assistant triggered a surge of other chatbots from big companies like *Google Now* in 2012, Amazon's *Alexa*, and Microsoft's *Cortana* in 2015. The main advantage of these chatbots is that they are strongly connected to many other software applications from the same parent company, allowing them to become true personal assistants.

In parallel to the success of these new personal assistants, end-to-end approaches to dialog systems started to be explored. In particular, Bengio, Ducharme, et al. (2003) developed a neural approach to the language modelling task. Given a set of tokens (words), the goal is to predict what is the next token following that sequence. This novel use of recurrent networks outperforms previous work based on n-gram features. Applied recursively this technique can potentially produce meaningful sentences.

What was missing to perform the dialog task was a way of conditioning that

generation process with the context of the conversation. A solution is proposed by Sutskever, Vinyals, and Q. V. Le (2014) that presents an encoder-decoder architecture, also known as a sequence-to-sequence model. The same recurrent network is used to first encode the conversation history into a fixed-length vector before generating the next possible sentence token by token as in the language modelling task.

Following this novel technique, researchers begun to explore data-driven generation of conversational responses in the form of encoder-decoder or sequence-to-sequence models (Sordoni et al., 2015; Vinyals and Q. Le, 2015; Serban, Sordoni, et al., 2016). However, the generated responses are often too generic to carry meaningful information, such as "I don't know.", which can serve as a response to any user questions. A mutual information based model was proposed by J. Li, Galley, Brockett, Gao, et al. (2015) to address this issue, and was later improved by using deep reinforcement learning (J. Li, Monroe, et al., 2016). Furthermore, J. Li, Galley, Brockett, Spithourakis, et al. (2016) presented a persona-based model to address the issue of speaker consistency in neural response generation.

Overall however, end-to-end systems are still not performing as well as modular architectures, and that is why we are proposing a novel type of chatbot in which we combine both modern end-to-end systems and simpler rule-based systems with deep learning ranking techniques, as described in Chapters 5 and 6.

2.2 History of ChatBot Competitions

In the previous section we described the most popular chatbots being developed. However, a lot of other conversational agents have been made over the years either for industrial purposes or as part of competitions. In an attempt to summarize previous work, we now present the competition that drove the field forward.

In 1990, Hugh Loebner and The Cambridge Centre for Behavioural Studies established a competition based on implementing the Turing Test. A Gold Medal and

$100,000 have been offered by Hugh Loebner as a Grand Prize for the first computer that makes responses which cannot be distinguished from humans. A bronze medal and an annual prize of $2,000 are pledged in every annual contest for the system that seems to be more human in relation to the other competitors. It is the first known competition that represents a formal Turing test (Epstein, 1993). The competition has been run from 1991 annually. The goal of this challenge is to design a chatbot that has the ability to pursue a conversation on any topic. The evaluation of the system is made by an interrogator that tries to guess whether they are talking to a program or a real human. After a five-minute conversation between the judge and a chatbot, and another five-minute conversation between the judge and an independent confederate, the judge has to nominate which one was the human. According to this judgment, the more human chatbot is the winner. No chatbot has ever won the golden medal and passed the test to win the Loebner Prize, that is, fooling all the judges. However, there is a winning bot every year that is able to fool at least a few of them. The chatbots that won are called Loebner Prized chatbots[2].

Through the years of Loebner Prize Competitions the winning chat technologies evolved from very simple pattern matching systems, towards complicated patterns in combination with knowledge bases (Bradeško and Mladenić, 2012). However, most systems are still strongly hand-crafted, and not a lot of automatic reasoning machine has been proposed.

Around the same time as the *ConvAI* challenge described in the previous chapter, another recent challenge was proposed by *Amazon*: the *Alexa Prize*[3]. This competition is targeted towards university students to advance human-computer interactions. The goal of the competition is to create a social chatbot that can converse coherently and engagingly with humans on a wide range of topics such as current events,

[2] https://en.wikipedia.org/wiki/LoebnerPrize
[3] https://developer.amazon.com/alexaprize

entertainment, sports, politics, technology, and fashion during 20 minutes. The submitted systems are evaluated by real Amazon users who already have an *Echo* device at their home. At any time, users can say something like "*Alexa, let's chat about* (a topic, for example, baseball playoffs, celebrity gossip, scientific breakthroughs, etc.)." In response, *Alexa* may direct the user to an anonymous team's chatbot to interact with, and the user can converse with the bot until he or she ends the conversation. At the end of the conversation, the users are asked to rate the conversational agent on a scale from 1 to 5 based on factors such as relevance, coherence and interestingness of conversations. The team with the highest average score wins. Ties are broken by average conversation lengths. This challenge is quite harder than the *ConvAI* challenge since no conversation topic is given to the chatbot before it starts interacting with real users. Furthermore, since *Alexa* is a voice-activated assistant, the chatbot relies on the accuracy of the speech recognizer provided. Many chatbots have been proposed for this challenge, overall they all rely on modern deep learning and reinforcement learning techniques and try to be as flexible as possible by avoiding following conversation rules[4]. Most notably the *MILABOT* (Serban, Sankar, et al., 2017) follows a similar structure as our *RLLChatBot* by first generating candidate responses before selecting one of them.

Both the *ConvAI* challenge and the *Alexa Prize* are running a second edition of the competition in 2018. These competitions provide a good alternative to the famous *Loebner Prize*, specifically for university labs that do not have the resources to compete against industrial teams. In addition, it is important to organize such challenges as automatic evaluation of open-domain dialog systems remains an open problem (C.-W. Liu et al., 2016). In particular the *ConvAI* challenge has released the dataset collected during the first human evaluation round, which is a first step towards understanding what makes a good conversation and what does not.

[4] https://developer.amazon.com/alexaprize/proceedings

3

Technical Preliminaries

> – It is literally the case that learning languages makes you smarter. The neural networks in the brain strengthen as a result of language learning.
>
> *Michael Gove*

In the previous chapter we reviewed past work in the chatbot community and the competitions to evaluate their performance. In this chapter we focus on the technical preliminaries that are used in this thesis and in general in the Deep Learning community. We start by defining the building blocks of an artificial neural network, we describe modern neural architectures used with sequential data, and eventually the algorithms to train them.

3.1 Artificial Neural Networks

Inspired by the human brain, Artificial Neural Networks (ANNs), also called *neural networks*, are data processing systems composed of many small units (McCulloch and Pitts, 1943; Hebb, 1949; Rosenblatt, 1958; Widrow and Hoff, 1960). Each of these units is called an *artificial neuron*. Usually, a lot of artificial neurons are connected to each other in a hierarchical layered structure. To mathematically model the firing rate

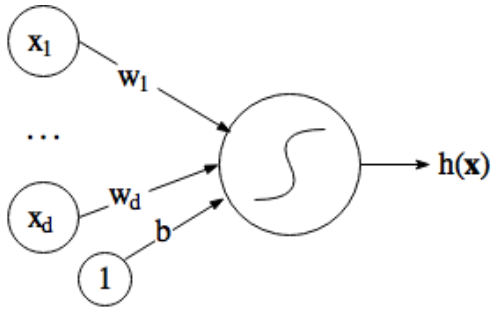


Figure 3.1: A graphical illustration of an artificial neuron. The input is the vector $\mathbf{x} = \{x_1, x_2, ..., x_d\}$ to which a weight vector $\mathbf{w} = \{w_1, w_2, ..., w_d\}$ and a bias term b is assigned. Each scalar is then summed and passed to an activation function.

of biological neurons, an *activation function* is used at the output of each neuron. In the next sub-sections we introduce the mathematical formulations of neural networks.

3.1.1 Artificial Neuron

An artificial neuron is a simple function from one or more inputs to a single output. Consider a set of inputs $\mathbf{x} = \{x_1, x_2, ..., x_d\}$ containing d input scalars. Each input is multiplied by a set of d scalar weights $\mathbf{w} = \{w_1, w_2, ..., w_d\}$ and added to a single scalar bias term b. After taking the sum of all these resulting scalars, we apply a non linear function called the *activation function*. A description of these steps can be seen in Figure 3.1. Formally, an artificial neuron $h(\mathbf{x})$ is defined as follows:

$$h(\mathbf{x}) = g(\sum_{i=1}^{d} w_i x_i + b), \tag{3.1}$$

in which $g(.)$ is the activation function. Consequently, we refer to the term $\sum_i w_i x_i + b$ as the *pre-activation*. For simplicity, the summation term can be written in vector multiplication:

$$h(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x} + b). \tag{3.2}$$

Note that for a single set of inputs, $\mathbf{x}$ is a vector of size $d \times 1$ while in the case of N sets of inputs, $\mathbf{x}$ is a matrix of size $d \times N$.

3.1.2 Activation Function

In Equation 3.1 the pre-activation is simply a linear weighted sum. In order to make the function from input to output nonlinear, the activation function $g(.)$ is applied on the pre-activation. Among different activation functions, here we introduce five popular ones.

Sigmoid (σ) activation function is an element-wise function that ranges between 0 and 1. The output of this function can be interpreted as the probability of a neuron being activated. Denoting the pre-activation as z, this function is defined as follows:

$$g(z) = \frac{1}{1 + exp(-z)}. \tag{3.3}$$

Hyperbolic Tangent (tanh) activation function is also an element-wise function but that ranges between -1 and 1. Thus, strongly negative inputs to tanh map to negative outputs and only zero-valued inputs are mapped to near-zero outputs. These properties let us have some amount of gradient information in the negative region, which can yield empirically better results. Denoting the pre-activation as z, this function is defined as follows:

$$g(z) = \frac{sinh(z)}{cosh(z)} = \frac{exp(z) - exp(-z)}{exp(z) + exp(-z)}. \tag{3.4}$$

Rectified Linear Unit (ReLU) (Glorot, Bordes, and Bengio, 2011) is also an element-wise function which, for negative inputs, is simply off (output is zero), while for positive inputs, the output is the same as the input. This activation function is one of the most common activation functions. Denoting the pre-activation as z, we can simply define this function as follows:

$$g(z) = \begin{cases} 0 & : z \leq 0 \\ z & : z > 0 \end{cases}. \tag{3.5}$$

Parametric Rectified Linear Unit (PReLU) (K. He et al., 2015) generalizes the rectified linear unit described above. For positive inputs, this function behaves

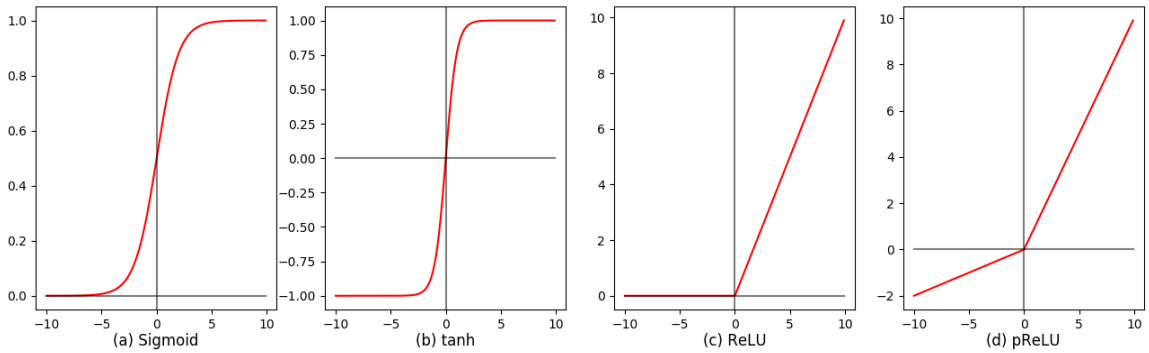


Figure 3.2: (a) the Sigmoid activation function, (b) the Hyperbolic Tangent activation function, (c) the Rectifier Linear Unit function, and (d) the pReLU activation function with $\alpha = 0.2$.

like the *ReLU* activation, but for negative examples, this function outputs negative values rather than 0. This avoids having zero gradient in the negative region. The slope of the negative portion of the function is a trainable parameter α. Denoting the pre-activation as z, we define this function as follows:

$$g(z) = \begin{cases} \alpha z & : z \leq 0 \\ \\ z & : z > 0 \end{cases}, \tag{3.6}$$

with α being the adjustable parameter taking values $\in [0, 1[$. Note that when $\alpha = 0$, *pReLU* is exactly the same as *ReLU*.

Softmax is another rather different type of activation function, which normalizes a set of pre-activations in such a way that each can be interpreted as a probability. Usually, in classification problems, if there are only two classes, a single Sigmoid unit is used. However, if there are more than two classes, the Softmax function is used. Considering $\mathbf{z}$ as a vector of C pre-activations, the Softmax over these C classes is defined as follows:

$$g(\mathbf{z})_j = \frac{exp(z_j)}{\sum_{i=1}^{C} exp(z_i)}. \tag{3.7}$$

A graphical depiction of the first four activation functions is shown in Figure 3.2. Note that since the Softmax is applied on more than two numbers, it is not as visualizable as the others.

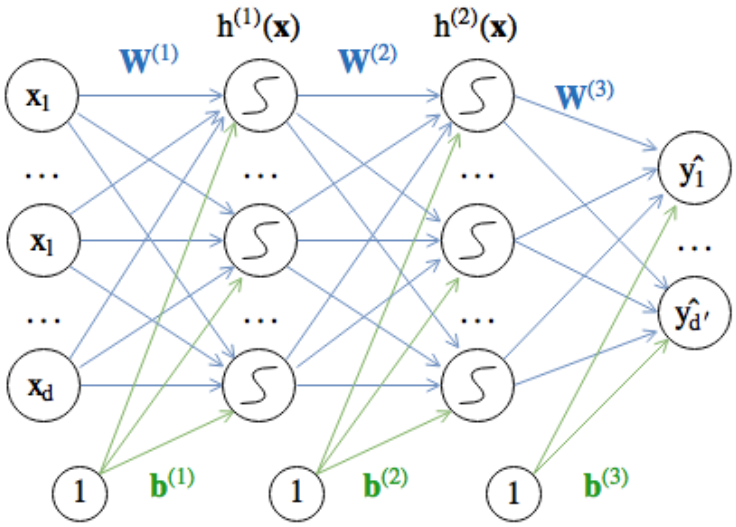


Figure 3.3: A three layer fully connected, feed-forward neural network. The matrix $\mathbf{W}^{(k)}$ connects the $(k-1)^{th}$ layer to the k^{th} layer and therefore $\mathbf{W}^{(k)} \in \mathbb{R}^{D^k \times D^{k-1}}$ and $\mathbf{b}^{(k)} \in \mathbb{R}^{D^k}$. After each linear transformation (weight multiplication and bias addition), an activation function is applied.

3.1.3 Multi-Layer Perceptron

A Multi-Layer Perceptron, also called *feed-forward* neural network (Levin and Fleisher, 1988; Hornik, Stinchcombe, and H. White, 1989) is a collection of many artificial neurons connected to each other in a layer-wise structure. As shown in Figure 3.3, in each individual layer, neurons with different weights and biases[1] are applied on all the inputs from the previous layer. Eventually, the output of that layer is the input for the next layer. Formally, a feed-forward neural network is defined recursively as follows:

$$\mathbf{h}^{(k)}(\mathbf{x}) = g(\mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x}) + \mathbf{b}^{(k)}), \tag{3.8}$$

$$\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}, \tag{3.9}$$

in which $\mathbf{W}^{(k)}$ is a matrix of real-valued parameters $\in \mathbb{R}^{D^k \times D^{k-1}}$, $\mathbf{b}^{(k)}$ is a vector of real-valued parameters $\in \mathbb{R}^{D^k}$, $\mathbf{h}^{(k)}(\mathbf{x})$ is the output of the k^{th} layer $\in \mathbb{R}^{D^k}$, and $g(.)$ is a differentiable[2], nonlinear activation function as described in Section 3.1.2. Usually,

[1] We refer to the weights $\mathbf{W}$'s and biases $\mathbf{b}$'s as parameters of the network.
[2] This is motivated in Section 3.3.3.

a single fixed activation function is chosen and used throughout the entire network.

The output layer of a neural network depends on the task. For regression tasks, where the goal is to predict a real-valued outcome, the final layer is often linear, omitting the activation function $g(.)$:

$$f(\mathbf{x}) = \mathbf{h}^{(n)}(\mathbf{x}) = \mathbf{W}^{(n)}\mathbf{x} + \mathbf{b}^{(n)}.$$

For classification tasks where each output neuron represents a class, a softmax distribution is often used for the last layer non-linearity, as described in Equation 3.7. The sum in the denominator is taken over all of the neurons in the output layer (i.e. over all classes). Thus, the output at each neuron represents the probability that the input $\mathbf{x}$ belongs to a certain class. As we see in Section 3.2, sequence-to-sequence problems in natural language are a specific instance of multi-step classification using recurrent neural networks.

To sum up, a multi-layer perceptron can be seen as a complicated function from inputs to outputs, composed of many small functions. According to the universal approximation theorem (Hornik, Stinchcombe, and H. White, 1989), multi-layer feed-forward networks are capable of approximating any measurable function to any desired degree of accuracy given sufficient number of layers and neurons per layer. However, finding the appropriate parameters $\mathbf{W}$'s and $\mathbf{b}$'s remains a challenging problem. In Section 3.3, we introduce the current methods for training such architectures.

3.2 Recurrent Neural Networks

In the previous section we introduced the multi-layer feed-forward neural network, universal function approximation, composed of many simple non-linear functions called artificial neurons. While being widely used, the multi-layer perceptron is not appropriate to encode sequential data because it has a fixed architecture and cannot be used for variable length inputs. This is a problem when working with conversational

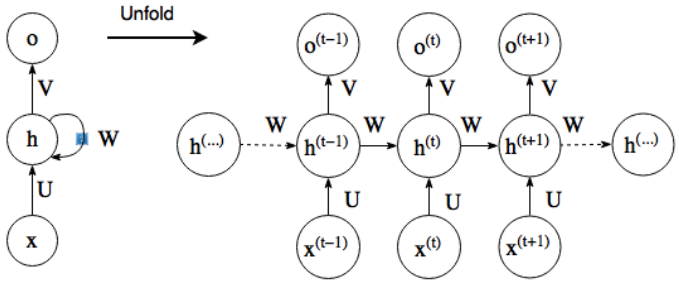


Figure 3.4: A computation graph of an RNN; the unrolled RNN is shown on the right. Matrices **U**, **W**, and **V** are the parameters of this network.

data as we define the length of sentences by their number of words. Thus, from one sentence to the next, the network architecture would need to be different in order to perform its computation on various inputs. In this section we introduce the Recurrent Neural Network, which solves that issue.

3.2.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) augment feed-forward neural networks with feedback loops. Most often, these are self-loops from hidden neurons in the same layer (Hopfield, 1982; Jordan, 1986; Elman, 1990). The idea is that we want to be able to relax the data-independence assumption. In other words, we want to model data that is correlated in time. This can take the form of sequences of input data $x_1, \cdots, x_m$. This is the case, for example in natural language, where each token x_i represents a word in some language. An RNN can be represented as a directed cyclic graph as illustrated in Figure 3.4.

The introduction of the feedback loop to the hidden unit of an MLP can be thought of as constructing a hidden state of the network $\mathbf{h}(\mathbf{x})$, that evolves over time as we present inputs to the network. The hidden state is updated at each time step according to some function f:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t). \tag{3.10}$$

It is important to note that the parameters of $f(.)$ are the same for all time steps.

This form of parameter sharing is crucial for RNNs: if we had different parameters for each time step, not only would the model be much more prone to overfitting, but it would not be able to generalize to sequence lengths unseen during training (Goodfellow et al., 2016).

More formally, a Recurrent Neural Network is defined by the following update equations at each time step t:

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}), \tag{3.11}$$

$$\mathbf{o}_t = softmax(\mathbf{V}\mathbf{h}_t + \mathbf{c}), \tag{3.12}$$

in which, $\mathbf{W}$ is the hidden-to-hidden matrix of parameters, $\mathbf{U}$ is the input-to-hidden matrix of parameters, $\mathbf{V}$ is the hidden-to-output matrix of parameters, $\mathbf{b}$ is the hidden state bias, and $\mathbf{c}$ is the output bias. The most common activation $g(.)$ used in RNNs is the *tanh* function (see Equation 3.4) because unlike the sigmoid activation, it does not bring values close to 0. In practice, the initial state h_0 can be initialized to a random vector of mean 0 and some positive standard deviation.

The configuration of RNNs can be modified to fit a wide variety of tasks. For example, an RNN with a single output at the last time step can be used to make a categorical prediction when taking sequential data as input, such as in sentiment analysis (Tang, Qin, and T. Liu, 2015). In addition, an RNN can also be used as a language model, where the output at each time step attempts to predict the next word in a sentence (Mikolov, Karafiát, et al., 2010). This mapping of an input sequence of words to itself is a simple kind of sequence-to-sequence problem; however, the current architecture is constrained by the fact that the length of the output sequence must be the same as the length of the input sequence. In Section 5.1.1, we show how to relax this assumption and allow variable-length inputs and outputs by having two RNNs.

Unfortunately, it has been observed that it is difficult to train RNNs to capture long-term dependencies because the gradients tend to either vanish (most of the time) or explode (rarely, but with severe effects) (Bengio, Frasconi, and Simard, 1993; Ben-

gio, Simard, and Frasconi, 1994). In the next subsections we introduce two gating mechanisms that aim to solve this issue.

3.2.2 Long Short-Term Memory Networks

Long short-term memory networks (LSTMs) are a special case of RNNs. They were introduced primarily in order to overcome the vanishing gradient problem (Hochreiter and Schmidhuber, 1997) with regular RNNs. LSTMs are explicitly designed to avoid the long-term dependency problem. All recurrent neural networks have the form of a chain of repeating modules of neural network when unrolled as we saw in Figure 3.4. LSTMs also have this chain like structure, but the repeating module itself (function $f(.,.)$ in Equation 3.10) has a different structure. Since the original paper, a lot of work has been done to improve the architecture of LSTMs in various tasks (Gers, Schmidhuber, and Cummins, 1999; Schuster and Paliwal, 1997; Gers and Schmidhuber, 2000; Zaremba and Sutskever, 2014; Sutskever, Vinyals, and Q. V. Le, 2014). In this section we describe the most common architecture that is used in the dialog research community.

The key idea behind LSTM networks is that it has a hidden state $\mathbf{h}_t \in \mathbb{R}^K$ (like in the RNN case) but also a memory state (also called *cell state*) that we denote $\mathbf{c}_t \in \mathbb{R}^K$. The hidden state can be though of as a *short-term* memory, while the memory state can be though of as a *long-term* memory. The memory state is only slightly modified through out the input sequence. It is thus easy for information to flow as the input gets longer. The memory state is carefully regulated by *gates*. Gates are a way to balance the amount of information that can be added to (or removed from) the cell state. Formally, if we define the previous LSTM state as the following tuple: $(\mathbf{c}_{t-1}, \mathbf{h}_{t-1})$ and the current input as $\mathbf{x}_t \in \mathbb{R}^N$, the gates at the current time

step are defined as follows:

$$\mathbf{f}_t = \sigma(\mathbf{W}^{fh}\mathbf{h}_{t-1} + \mathbf{W}^{fx}\mathbf{x}_t + \mathbf{b}^f), \tag{3.13}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}^{ih}\mathbf{h}_{t-1} + \mathbf{W}^{ix}\mathbf{x}_t + \mathbf{b}^i), \tag{3.14}$$

$$\hat{\mathbf{c}}_t = tanh(\mathbf{W}^{ch}\mathbf{h}_{t-1} + \mathbf{W}^{cx}\mathbf{x}_t + \mathbf{b}^c), \tag{3.15}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^{oh}\mathbf{h}_{t-1} + \mathbf{W}^{ox}\mathbf{x}_t + \mathbf{b}^o), \tag{3.16}$$

where we denote $\mathbf{f}_t \in \mathbb{R}^K$ as the *forget gate* taking values between 0 and 1, $\mathbf{i}_t \in \mathbb{R}^K$ as the *input gate* taking values between 0 and 1, $\hat{\mathbf{c}}_t \in \mathbb{R}^K$ as a vector of new candidate values between -1 and 1 for $\mathbf{c}_t$, and $\mathbf{o}_t \in \mathbb{R}^K$ as the *output gate* taking values between 0 and 1. Note that for each equation above, we have two weight matrices ($\mathbf{W}^{\cdot h} \in \mathbb{R}^{K\times K}$, $\mathbf{W}^{\cdot x} \in \mathbb{R}^{K\times N}$) and one bias vector ($\mathbf{b}^{\cdot} \in \mathbb{R}^K$) as parameters. Just like in the classical RNN case, those eight matrices and four vectors are the same for all time steps in the input sequence.

After computing the three gates and the candidate vector, the update to the LSTM state is defined according to the following equations:

$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \hat{\mathbf{c}}_t, \tag{3.17}$$

$$\mathbf{h}_t = \mathbf{o}_t * tanh(\mathbf{c}_t). \tag{3.18}$$

From this, we can interpret the *forget gate* as a way of measuring how much of the previous memory state we keep, the *input gate* measures how much of the candidate state we keep, and the *output gate* measures how much of the new memory cell we need to output at this moment.

As mentioned above, simple RNNs may fail to capture long term dependencies, and LSTMs aim to solve this issue. As a result LSTMs are being used in a wide variety of tasks where we need to capture time dependence in the input. In the next subsection we describe another (more recent) gated mechanism that has now become popular in the community.

3.2.3 Gated Recurrent Unit Networks

Gated recurrent unit networks (GRUs) are a simplified version of LSTMs. This architecture was motivated by the success of LSTMs, but it is much simpler to compute and implement (Cho et al., 2014a). As before, the GRU has the same unfolded layout shown in Figure 3.4, but a different mechanism to compute the hidden state $\mathbf{h}_t$ at each time step; in other words, we simply modify the function f in Equation 3.10.

The main difference with LSTMs is that GRUs, just like classic RNNs, keep track of only one state vector over time: $\mathbf{h}_t$. The update on this *hidden* state is regulated by 2 gates and one candidate vector. More formally, if we define the previous GRU state as $\mathbf{h}_{t-1} \in \mathbb{R}^K$ and the current input as $\mathbf{x}_t \in \mathbb{R}^N$, the gates at the current time step are defined as follows:

$$\mathbf{r}_t = \sigma(\mathbf{W}^{rh}\mathbf{h}_{t-1} + \mathbf{W}^{rx}\mathbf{x}_t), \tag{3.19}$$

$$\hat{\mathbf{h}}_t = tanh(\mathbf{W}^{hh}(\mathbf{r}_t * \mathbf{h}_{t-1}) + \mathbf{W}^{hx}\mathbf{x}_t), \tag{3.20}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}^{zh}\mathbf{h}_{t-1} + \mathbf{W}^{zx}\mathbf{x}_t), \tag{3.21}$$

where $\mathbf{r}_t \in \mathbb{R}^K$ is the *reset gate* taking values between 0 and 1, $\hat{\mathbf{h}}_t \in \mathbb{R}^K$ is a vector of new candidate values between -1 and 1 for $\mathbf{h}_t$, and $\mathbf{z}_t \in \mathbb{R}^K$ is the *update gate* taking values between 0 and 1. Note that in this setting there are only 3 pairs of matrices ($\mathbf{W}^{\cdot h} \in \mathbb{R}^{K\times K}$, $\mathbf{W}^{\cdot x} \in \mathbb{R}^{K\times N}$) of parameters to optimize. We can interpret the reset gate as measuring the amount of information to keep from the previous hidden state to the candidate state, thus allowing the model to forget any information that is found to be irrelevant in the future.

Eventually, the new hidden state in a GRU is computed as follows:

$$\mathbf{h}_t = \mathbf{z}_t\mathbf{h}_{t-1} + (1 - \mathbf{z}_t)\hat{\mathbf{h}}_t. \tag{3.22}$$

Here we can see that the update gate controls how much information from the previous hidden state is carried over to the current state. This behaves similarly to the memory state in the LSTM network and helps the RNN to remember longterm information.

It has been shown empirically that GRU networks can yield similar results than LSTMs while being simpler to optimize due to the smaller number of parameters (Cho et al., 2014a; Cho, Van Merriënboer, Bahdanau, et al., 2014; Chung et al., 2014). As such GRUs have quickly became the preferred alternative to LSTMs.

3.3 Training Artificial Neural Networks

In the two previous sections, we introduced various structures of artificial neural networks. We also defined two sets of parameters: weights and biases. In order for the network to model a specific mapping (or function) from inputs to outputs, its parameters need to be updated. The process of adapting the parameters is called *training*. In the following subsections, we introduce methods developed to optimize the parameters with respect to a predefined loss function.

3.3.1 Cost Functions

In optimization problems, the *cost function* (or *loss function*) is a function from a set of values (the network parameters here) to a single real number. This real number is called the "cost". The objective of neural network training procedures is to minimize the cost between the current output of the network and the required output.

To design an empirical cost function, consider a network with parameters θ that map input $\mathbf{x}$ to output $\hat{\mathbf{y}}$. The objective is to minimize the loss between the current output $\hat{\mathbf{y}}$ and the targeted output $\mathbf{y}$. Having N pairs of $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ and $\hat{\mathbf{y}}^{(i)} = f_\theta(\mathbf{x}^{(i)})$, we define a generic loss function:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_i l(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}), \tag{3.23}$$

in which $l(.,.)$ is a differentiable function that measures the error we make on each prediction, and $\mathbf{x}^{(i)}$, $\mathbf{y}^{(i)}$ are input, output vectors respectively. This setting, where we have N example pairs of (input, output), is called *Supervised Learning*. There are many distinct cost functions that can be used to train neural networks in a supervised manner. The choice mostly depends on the task at hand. Here we introduce four popular cost functions.

Negative Log-Likelihood (NLL) is often used in classification tasks with C classes, the value of each output neuron is interpreted as the probability of the input belonging to that specific class, i.e., $\hat{\mathbf{y}}^{(i)}[c] = Pr(\mathbf{y}^{(i)} = \mathbf{c}|\mathbf{x}^{(i)})$. Similarly, NLL is used in sequence prediction tasks where the value of each output neuron (in an RNN) is a conditional probability, i.e., $h_t = Pr(y_t^{(i)}|y_1^{(i)}, \cdots, y_{t-1}^{(i)}, x_t^{(i)})$. We measure the log probability of the output sentence and try to maximize it. Note that this scenario can be considered as a large classification task where C is the size of our vocabulary, and we try to maximize the probability of the correct token at each time step. In general, the negative log-likelihood function is defined as follows:

$$l(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = -\sum_{c \in C} 1_{[y^{(i)}=c]} \log \hat{\mathbf{y}}^{(i)}[c] = -\log \hat{\mathbf{y}}^{(i)}[y^{(i)}], \tag{3.24}$$

in which $1_{[.]}$ is the indicator function, $\hat{\mathbf{y}}^{(i)}$ is the predicted vector of probabilities, $\mathbf{y}^{(i)}$ is the one-hot vector representing target class $y^{(i)}$, c is a class index, and the log is used for numerical stability and also for mathematical simplicity.

Cross Entropy (CE) loss is also often used in classification tasks as it is similar to the NLL loss. The cross entropy loss applies a softmax activation function (as described in Equation 3.7) on the output layer before applying the NLL loss. Denoting the output layer as a vector $\hat{\mathbf{y}}^{(i)} \in \mathbb{R}^C$, where C is the number of classes (or vocabulary size in sequence prediction), the function is defined as follows:

$$l(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = -\log(\frac{exp(\hat{\mathbf{y}}^{(i)}[y^{(i)}])}{\sum_{j \in C} exp(\hat{\mathbf{y}}^{(i)}[j])}), \tag{3.25}$$

where $\hat{\mathbf{y}}^{(i)}$ is the predicted vector of probabilities, $\mathbf{y}^{(i)}$ is the one-hot vector representing target class $y^{(i)}$ and j is any class index.

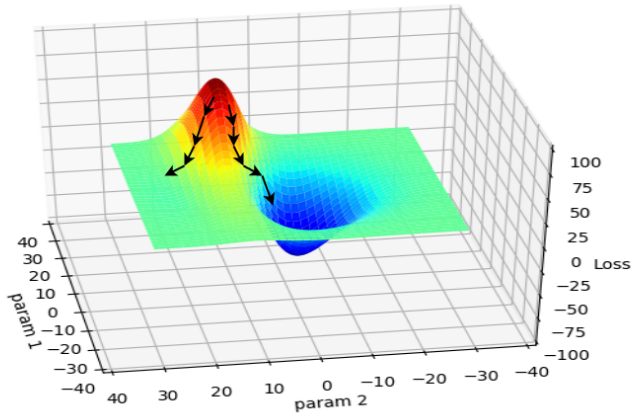


Figure 3.5: A graphical description of the gradient descent algorithm. Here we consider the case where we have only two parameters and we plot on the z axis the value of the cost function.

Mean Squared Error (MSE) is often used in regression tasks where the target $\mathbf{y}$ is a vector of real values. Denoting the network predictions as $\hat{\mathbf{y}}^{(i)} = f_\theta(\mathbf{x}^{(i)})$, the function is defined as follows:

$$l(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = (\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)})^2. \tag{3.26}$$

Huber loss is similar to the MSE loss, but it is less sensitive to outliers and in some cases prevents exploding gradients (Girshick, 2015). When the difference between the predicted scalar and the target is small (less than 1) the MSE loss is applied; otherwise, the absolute difference is used (also called the L1 loss). Denoting the network predictions as $\hat{\mathbf{y}}^{(i)} = f_\theta(\mathbf{x}^{(i)})$, the function is defined as follows:

$$l(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = \begin{cases} 0.5(\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)})^2 & : |\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}| < 1 \\ |\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}| - 0.5 & : o.w. \end{cases}. \tag{3.27}$$

3.3.2 Gradient Based Optimization

In order to optimize a loss function, we must set its derivative to zero and solve with respect to its parameters. However, in practice this equation has often no solution or is impossible to compute. Thus, artificial neural networks are typically trained

using gradient based optimization methods, and specifically, using the Gradient Descent algorithm and its variants (Rumelhart, G. E. Hinton, and R. J. Williams, 1985; Rumelhart, G. E. Hinton, and R. J. Williams, 1986). Gradient Descent finds a local minimum of a function by taking small steps in the direction of the gradient proportional to its magnitude. Therefore, gradient descent is an iterative process that at each iteration updates the parameters using the following update rule:

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_{\theta_t} L(\theta_t), \tag{3.28}$$

where α is called the *learning rate*.

In other words, the gradient descent algorithm can be seen as a hiker climbing down a hill to a part with lowest height. Each step of the hiker is determined by the slope of the hill at that specific location. A graphical description of the gradient descent algorithm is shown in Figure 3.5.

Typically, *Stochastic (mini-batch) Gradient Descent* (SGD) is used rather than Gradient Descent. SGD is a version of Gradient Descent in which instead of computing the gradient $\nabla_\theta L(\theta)$ exactly, an estimate of the gradient is used based on a few randomly selected examples (called a *batch* of data). Since the examples are randomly selected, the expected value of the gradient is the same as the exact gradient.

In practice, the learning rate is an important hyper-parameter that can significantly affect learning. One way to handle this problem is to use learning rates which can adapt throughout the course of learning. In recent years, several algorithms with adaptive learning rate have been proposed such as *Adagrad* (Duchi, Hazan, and Singer, 2011), *Adadelta* (Zeiler, 2012), and *Adam* (Kingma and Ba, 2014). For the most part, these algorithms rely on momentum to adapt the learning rate. In practice we find that such optimization algorithms yield better training behaviors than conventional SGD.

Due to the nonlinearity of the activation function $g(.)$, gradient descent for neural networks is often non-convex in the parameters. While theoretically this provides

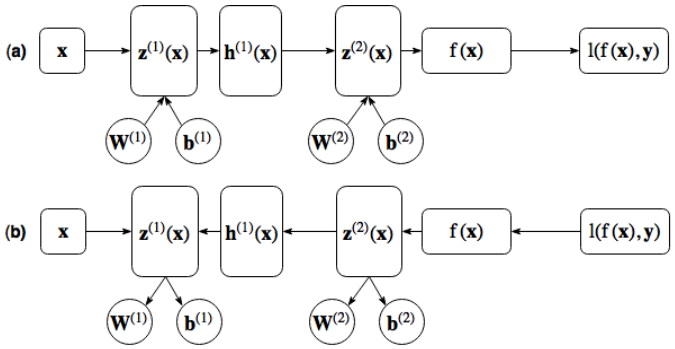


Figure 3.6: (a) The forward pass and (b) the backward pass in the backpropagation algorithm on a two-layer neural network. Consistent with our previous notation, we define $\mathbf{z}^{(l)}$ and $\mathbf{h}^{(l)}$ as the pre-activation and the activation at layer l respectively. Note that in the backward pass, each module needs to be differentiable.

challenges for analyzing neural networks, in practice neural networks work well despite this limitation.

3.3.3 Backpropagation Algorithm

In the previous sections, we described how to train neural network parameters using gradients of their parameters. In this section, we introduce a well-known algorithm for computing the gradients in an efficient way. In a multi-layer network, backpropagation (Rumelhart, G. E. Hinton, and R. J. Williams, 1986) uses the chain rule to iteratively compute the gradients. It is important to note that in order to be able to use backpropagation, both activations and pre-activations must be differentiable.

Each step of the backpropagation algorithm contains one forward and one backward pass. The forward pass consists of feeding inputs to the network, predicting output values, and eventually computing the error (the cost function) between the predicted and the target values. As indicated by its name, the backward pass goes in the opposite direction: it takes the error at the output layer previously computed and propagate it to each neuron parameters. Figure 3.6 depicts the forward and backward paths using a flow graph.

Let us consider the network in Figure 3.6. Formally, the first step of the back-

ward pass is to compute the gradient of the loss function with respect to the pre-activation at layer 2: $\nabla_{\mathbf{z}^{(2)}} l(f(\mathbf{x}), \mathbf{y})$. Using chain rule, since $\nabla_{\mathbf{W}^{(2)}} \mathbf{z}^{(2)}(\mathbf{x}) = \mathbf{h}^{(1)}(\mathbf{x})$ and $\nabla_{\mathbf{b}^{(2)}} \mathbf{z}^{(2)}(\mathbf{x}) = \mathbf{1}$, the next step is to compute the gradients with respect to each parameters:

$$\nabla_{\mathbf{W}^{(2)}} l(f(\mathbf{x}), \mathbf{y}) = \nabla_{z^{(2)}(\mathbf{x})} l(f(\mathbf{x}), \mathbf{y}) \mathbf{h}^{(1)}(\mathbf{x})^T, \tag{3.29}$$

$$\nabla_{\mathbf{b}^{(2)}} l(f(\mathbf{x}), \mathbf{y}) = \nabla_{z^{(2)}(\mathbf{x})} l(f(\mathbf{x}), \mathbf{y}). \tag{3.30}$$

Eventually, to back-propagate the gradient to the previous layer, we have:

$$\nabla_{\mathbf{h}^{(1)}} l(f(\mathbf{x}), \mathbf{y}) = \mathbf{W}^{(2)^T} \nabla_{\mathbf{z}^{(2)}(\mathbf{x})} l(f(\mathbf{x}), \mathbf{y}), \tag{3.31}$$

$$\nabla_{\mathbf{z}^{(1)}} l(f(\mathbf{x}), \mathbf{y}) = \nabla_{\mathbf{h}^{(1)}(\mathbf{x})} l(f(\mathbf{x}), \mathbf{y}) \odot g'(\mathbf{z}^{(1)}(\mathbf{x})), \tag{3.32}$$

in which $\odot$ is an element-wise multiplication and $g'(.)$ is the derivative of the activation function. Now that we have $\nabla_{\mathbf{z}^{(1)}(\mathbf{x})} l(f(\mathbf{x}), \mathbf{y})$, gradients of parameters $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ can be computed in a similar fashion.

In order to train RNNs, *backpropagation through time* (BPTT) is used. The algorithm is precisely the same as the usual backpropagation algorithm, but is applied on the unrolled RNN graph as seen in Figure 3.4.

Published more than 30 years ago, the backpropagation algorithm is still at the center of every learning procedure in Deep Learning today (Glorot and Bengio, 2010; Lipton, Berkowitz, and Elkan, 2015; C.-W. Liu et al., 2016). Being flexible, this algorithm can be applied to any kind of architecture as long as each component of the model, including the loss function is differentiable.

3.4 Reinforcement Learning

Introduction

In the previous subsection we described a well-known algorithm to train neural networks that need to minimize a cost function (or maximize its opposite). However,

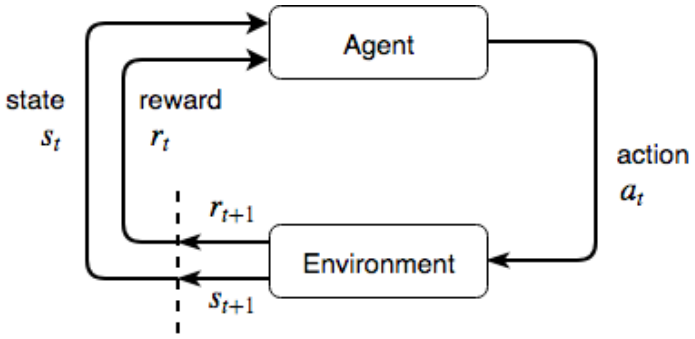


Figure 3.7: Typical reinforcement learning loop between an agent and the environment it is in. At each time step t, the agent performs some action a_t on the environment based on its current state s_t, observe a reward r_t and moves to a next state s_{t+1}.

we assumed that the loss function was differentiable. In this section we relax this assumption and consider the case in which we want to optimize parameters according to a non-differentiable function. In *Reinforcement Learning* (RL) it is often the case to think about a *reward* function that we want to maximize rather than a cost function to minimize (Bellman, 1957; Watkins and Dayan, 1992; Littman, 1994; Sutton and Barto, 1998). We now describe the high level overview of this framework.

Reinforcement Learning is inspired by how most living beings learn their environment: they interact with the world, observe rewards and adapt their behavior as such. Similarly, in RL we let the system act in a simulated world and give it a positive or negative reward after one or more actions. The goal of the agent is simply to maximize the sum of its future discounted rewards (*the return*): $R = \sum_{t=0}^{T} \gamma^t r_t$, where γ is a discount factor between 0 and 1 and T is the time horizon. In other words, we let the agent learn by trial and error where at every time step it takes an action a in the environment based on its current perception (defined as state s), observe a reward r and moves to a next state s'. A pictorial representation of this mechanism is shown in Figure 3.7.

3.4.1 Markov Decision Process

In order to formally define the reinforcement learning problem, let us start by defining a Markov Decision Process (MDP). An MDP (Bellman, 1957) is a tuple $< S, A, T, r, \gamma >$ consisting of a set of states S, a set of actions A, a set of transition probability distribution $T : A \times S \to (S \to [0, 1])$ that determines the probability of being in a subsequent state given the previous state and action ($P(s'|s, a) \in [0, 1]$), a reward function $r : A \times S \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1]$. Agents acting in an MDP use either a stochastic policy $\pi : A \times S \to [0, 1]$ or a deterministic policy $\pi : S \to A$ in order to act in the environment while trying to maximize their return R. Reinforcement learning methods are an approach to finding the best policy ($\pi^\star$) that maximizes the return R. The environment dynamics T may be known but in practice, they are often unknown and we only have sample-based access to the environment.

In our chatbot setting, we consider a 2-agent extension of Markov decision processes called partially observable Markov games (Littman, 1994). The partial observability comes from the fact that we cannot be sure that we understood what the human user said. Thus we model this uncertainty by having a partially observable environment. We now describe a popular RL algorithm for tackling MDPs and Markov games.

3.4.2 Q-learning and Fitted Q-Iteration

Q-learning is a popular method in reinforcement learning making use of an *action-value function* $Q^\pi(s, a)$ that gives an estimate of the expected future reward that can be obtained by taking action a in state s, and following the policy π afterwards (Sutton and Barto, 1998). More formally we have: $Q^\pi(s, a) = \mathbb{E}_\pi[R|s_t = s, a_t = a]$. To simplify notation a little, let us define $s = s_t \in S$, $s' = s_{t+1} \in S$, $a = a_t \in A$, and $a' = a_{t+1} \in A$. Using the *Bellman equation* (Bellman, 1957), we can rewrite the Q

function recursively like so:

$$Q^{\pi}(s,a) = \mathbb{E}_{s'}[r(s,a) + \gamma \mathbb{E}_{a' \sim \pi}[Q^{\pi}(s',a')]] \tag{3.33}$$

$$= \mathbb{E}_{s'}[r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^{\pi}(s',a')]. \tag{3.34}$$

The optimal Q-function ($Q^{\star}$) can be determined by solving the above equation. Eventually, the optimal policy ($\pi^{\star}$) is then $\pi^{\star}(s) = \arg\max_{a \in A} Q^{\star}(s,a)$.

However, the model of the environment ($r(s,a)$ and $P(s'|s,a)$) has to be known in order to solve this recursive equation. When no such information is available we need to estimate the environment by acting in it. One way of doing so is by running a form of *experience replay* (L.-J. Lin, 1992) algorithm called *fitted value iteration* (Gordon, 1995) where value iteration is performed on experiences seen so far. This is what inspired others to develop *Fitted Q-Iteration* (Ernst, Geurts, and Wehenkel, 2005) and *Neural Fitted Q-Iteration* (NFQI - Riedmiller, 2005) algorithms to directly estimate the Q-function by using tree based regression methods and multi-layer perceptrons respectively. These algorithm rely on previous experience to learn an approximation of the Q-function. It is considered to be a case of *batch* or *off-line* Reinforcement Learning as we are estimating a Q-value with batches of data coming from a fixed dataset, and the agent is not allowed to interact in the environment during training. This can be viewed as a special case of *supervised learning* where instead of predicting a label, we try to imitate a (presumably expert) behavior by estimating a state-value action. Moreover, because the policy that is estimated is the same as the policy used to collect the trajectories (i.e. we want to reproduce the same expert policy) this technique is a case of *on-policy* algorithm.

In classical Q-learning methods however, we do not have access to expert behaviors and we are forced to let the agent explore its environment until it finds (by itself) the best policy to maximize its return (Sutton and Barto, 1998). In contrast to *off-line* exploration, this is called *online* exploration. In addition, here the model explores the environment by making use of an ϵ-greedy policy, where the agent chooses a

random action with probability ϵ (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013). Q-learning is an *off-policy* learning algorithm: unlike NFQI, the policy that we estimate is different than the one used to collect the trajectories (in particular, we directly try to estimate the optimal policy, while the policy used to collect the trajectories is ϵ-greedy). From one side, we can imagine that the model may take more time to learn by itself, but it has been shown in a recent derivative of Q-learning (Deep Q-Network or DQN) that letting the agent act in the environment can actually outperform human accuracy in some game-playing tasks (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013; Wang et al., 2015; Van Hasselt, Guez, and Silver, 2016a). In DQN, we learn an approximation of this Q-function using a deep neural network that outputs a set of action values $Q_\phi(s, .)$ for a given state input s, where ϕ are the parameters of the network. In order to learn those parameters we minimize this MSE loss (see Equation 3.26):

$$\mathcal{L}(\phi) = \mathbb{E}_{s,a,s'}[(Q^\pi_\phi(s,a) - y)^2], \quad \text{where} \quad y = r(s,a) + \gamma \max_{a'} Q^\pi_{\bar{\phi}}(s', a'), \tag{3.35}$$

where $Q^\pi_{\bar{\phi}}$ is a target Q-function that uses old parameters $\bar{\phi}$ from earlier in training, which helps stabilize learning. Indeed, if the most recent Q^π_ϕ was used as target instead in the MSE loss, the network would be chasing a non-stationary target and may never converge. These 'old' parameters are periodically updated with the most recent ϕ. More recently, the double Q-learning update was proposed to reduce the upward bias that is created with regular Q-learning updates (Van Hasselt, Guez, and Silver, 2016b). The trick is to use the *current* network to calculate the argmax over next state-action values and the *target* network for the value of that action. Formally, the double DQN loss is the same as in Equation 3.35 but defines the target y like so:

$$y = r(s,a) + \gamma Q^\pi_{\bar{\phi}}(s', \arg\max_{a'} Q^\pi_\phi(s', a')), \tag{3.36}$$

where $\bar{\phi}$ are the parameters of the *target* network, and ϕ are the parameters of the *current* network. The difference is that instead of taking the action with maximal

Q-value according to the target network, we take the action with maximal Q-value according to the current network, and then measure that action's value according to the target network. This technique gives a better estimate of the true target y we try to reach (Van Hasselt, Guez, and Silver, 2016b).

Another crucial component of stabilizing the training of DQNs is the use of an experience replay buffer containing transition tuples (s, a, r, s') just like the set of expert behaviors in NFQI. This is because when the agent is performing actions in the environment, the set of transitions are correlated within the same trajectory, thus making transition data non-independent. However, in our case we didn't train the Q-network against a live environment for practical reasons described in Chapter 6. Instead we collected a large dataset of (s, a, r, s') transitions from human users[3], and trained the DQN to mimic those human transitions, considered to be optimal. We are thus doing '*Deep Neural Fitted Q-Iteration*'.

It is important to remember that in this entire section we did not assume the reward function r to be differentiable, which makes RL an appealing setting in many applications such as games (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013; Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015; Van Hasselt, Guez, and Silver, 2016b). In Chapter 6, we show how we used a DQN for our chatbot.

[3]Data collection procedure described in Chapter 7.

4

Data-Driven Dialog Systems

> – Un dialogue, ce n'est pas des mots, c'est un sens. Un dialogue doit dire quelque chose qui est l'action, la raison pour laquelle le dialogue est dit.
> – A dialogue is not words, it is a meaning. A dialogue must say something that is the action, the reason why the dialogue is said.
>
> *René Clément in an interview from 'France Culture'*

In the previous chapter we addressed several technical preliminaries regarding different types of neural networks and how to train them. In this chapter we introduce the four major components that together make a dialog system: Natural Language Interpreter, Dialog State Tracker, Dialog Manager, and Natural Language Generator (Rudnicky et al., 1999; Zue and Glass, 2000; S. J. Young, 2000). Each component is trained separately and either takes a more structured input (such as a set of dialogue acts), or is trained to maximize an intermediary objective (such as slot-filling). We call this pipeline a 'modular' approach to dialogue systems. A pictorial representation can be seen in Figure 4.1. Note that for this work we ignore the modules connected to audio signals (speech recognizer & speech synthesizer) since we focus only on text-to-text interactions.

More formally, we define a modular dialogue system as a system where elements (sub-components or system parameters) are optimized with respect to different objective functions. For instance, as we will see below, the natural language interpreter is

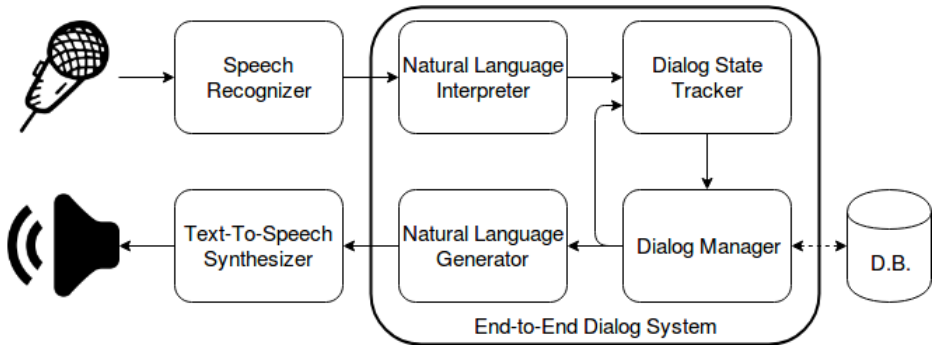


Figure 4.1: Pipeline framework of modular dialog systems. Composed of: a Speech Recognizer to translate audio signals to text, a Natural Language Interpreter that explains what the system heard by labeling the text, a Dialog State Tracker to understand what the user wants, a Dialog Manager to perform the required action and return some information, a Natural Language Generator to make a syntactical sentence, and a Text-To-Speech Synthesizer to map text to audio signals. In addition, an external database is often also communicating with the Dialog Manager.

trained on a classification task, while the dialog manager is trained in a reinforcement learning setting with the slot-value pairs output from the state tracker as a dialog state. Modular dialogue systems have been historically preferred over end-to-end systems (Callejas and López-Cózar, 2005; Raux et al., 2005; S. Young et al., 2013a). This is because such modular systems are easier to train, require less data, and so far have been shown to achieve better results in practice, albeit typically for highly structured tasks. It is also easier to manually program each component specifically to obey certain task constraints or to solve one or more isolated tasks.

4.1 Natural Language Interpreter

The role of the Natural Language Interpreter is to do Natural Language Understanding (NLU), which consists of domain identification and intent prediction (framed as utterance classification problems), and slot filling (framed as a sequence tagging task).

4.1.1 Utterance Classification

Given a collection of n utterances $u_{1,...,n}$ with labels[1] $c_{1,...,n}$, the goal is to train a model that estimates labels for new utterances $u_{n+1,...}$. As seen in Section 3.3.1, this is a case of *Supervised Learning*.

With the advances on deep learning, recent development has been focused on neural approaches. One typical way of doing so is to represent each utterance as a float vector and map this vector to a probability for each label. This can be done using Recurrent Neural Networks as described in Section 3.2.1 where each time step is considered as a word in the utterance, and the final output is a *Softmax* activation (Equation 3.7). We also represent words by float vectors known as *word embeddings* or simply *word vectors* (Mikolov, K. Chen, et al., 2013; Mikolov, Sutskever, et al., 2013; Pennington, Socher, and Manning, 2014).

One way of performing utterance classification is by predicting the target class probabilities after each token in the encoding RNN as in Ravuri and Stolcke (2015). Another solution is to make the decision based on the conversation history. Indeed, user utterances can be highly ambiguous in isolation (ex: *"book a table at <restaurant> for 6"* 6 may refer to the time or to the number of people). At every time step in the conversation, previous utterances can provide information about the user's intent (J. Y. Lee and Dernoncourt, 2016). Eventually, trying to understand the user's intention is an important aspect of the Language Interpreter module as it will help other modules generate appropriate responses.

4.1.2 Sequence Tagging

Another important task that the Natural Language Interpreter should do is to tag each token in an utterance. A tag can be a part-of-speech tag (such as *'NN'* for noun, *'JJ'* for adjective, *'V'* for verb, etc...) or a named-entity tag (such as *'PERSON'*,

[1] Usually the label represents the user's intent from a list of predefined intents.

'LOCATION', *'TIME'*, etc...). Given a collection of tagged word sequences $D = \{((w_{11}, t_{11}), (w_{12}, t_{12}), \ldots, (w_{1n}, t_{1n})), (\ldots)\}$, the goal is to estimate tags for a new word sequence.

As in the sequence classification task, *word embeddings*, RNN and LSTM networks are the architectures of choice where we try to predict the tag of each token at each time step in the recurrent network (Mesnil et al., 2015). Another solution to this problem has been to first encode the entire utterance in a fixed length vector, and then decode with another RNN the tags of each input tokens. In some cases it is easier to perform the slot-filling task after seeing the full utterance (Kurata et al., 2016). Once a sentence has been tagged with part-of-speech or entities, it can be used by the dialog state tracker to better understand the user's intent and thus the state of the dialog. Furthermore, modern approaches saw that identifying both the utterance tags for each token and the overall intent of the same utterance at the same time can yield better performance in each of the task (Hakkani-Tür et al., 2016; B. Liu and Lane, 2016).

Eventually, the evaluation of NLU systems is often based on intent accuracy and slot F1 scores. The importance of the NLU module is investigated in X. Li, Y.-N. Chen, L. Li, Gao, and Celikyilmaz (2017), showing that different types of errors from the Natural Language Interpreter can degrade the whole system's performance in a reinforcement learning setting.

4.2 Dialog State Tracker

After recognizing what the user said with the NLU module, it is important to understand what the user wants and where we are in the conversation in order to better respond. The goal of the Dialog State Tracker is to maintain a probabilistic distribution over the different states in which the dialog can be, it keeps a full representation

of the system's belief of the user's goal at any point during the dialog. From this, the dialog manager will then be able to make API calls to external databases in order to give appropriate information to the user.

In practice, the state tracker module receives as input the NLU module output (tagged sentences and predicted user's intent) as conversation history and predicts the current *dialog state*. In general, the dialog state might indicate the user's preference about some product, what information they are seeking and which concepts have been stated or confirmed. This task is very important because it provides the dialog manager with the current state of dialog that will influence what action is taken (i.e. what information to return).

In 2013, researchers from different affiliations decided to come together and propose a dialog state tracking challenge (DSTC) in order to push the boundaries forward (J. Williams, Raux, Ramachandran, et al., 2013). Since then, there is a DSTC organized almost every year and each time the focus is a little different, going from determining changes in the user goal, adapting to new domains or new languages (Henderson, Thomson, and J. D. Williams, 2014a; Henderson, Thomson, and J. D. Williams, 2014b; S. Kim, D'Haro, et al., 2017; S. Kim, D'Haro, et al., 2016).

In general, the best state trackers use some neural architecture to monitor dialogue progress (state). The first step is to extract relevant features from the source text and represent them in a vector. This can be done manually (Henderson, Thomson, and S. Young, 2013) or automatically by extracting n-grams representations (Mrkšić et al., 2016). If there is sufficient training data, this step can also be done using recurrent networks or convolutional networks (B.-J. Lee and K.-E. Kim, 2016). The second step is to predict the appropriate dialog state given the conversation seen so far. The state is usually represented as a list of slot-value pairs where a slot can be for instance *"food type"* or *"price range"* and a value can be *"Italian"* or *"expensive"* (J. Williams, Raux, Ramachandran, et al., 2013; Henderson, Thomson, and J. D. Williams, 2014a). For this step multiple layers of non-linear activation functions (i.e. feed-forward neural

network) are often used to predict the score of all possible candidate pairs (Henderson, Thomson, and S. Young, 2013; Mrkšić et al., 2016).

Eventually, the state tracker is evaluated on its tracked state accuracy, its recall, precision, and F-measure of states. In practice though, it is trained to minimize the cross-entropy error of predicting slot-value pairs (Henderson, Thomson, and S. Young, 2013; Mrkšić et al., 2016).

4.3 Dialog Manager

After identifying in which state the dialog is, the Dialog Manager is responsible for selecting the appropriate action (response) to the user. This is often framed as a reinforcement learning problem where the system has to pick the action that will maximize the sum of future expected rewards. Defining a proper reward depends on the task of the chatbot and is in general a difficult thing to do. In goal-oriented systems the rewards can be to maximize the success rate, while minimizing the number of turns. But in open-domain settings, the best reward may come from human users' ratings which are time-consuming and expensive to acquire (C.-W. Liu et al., 2016).

A partially observable Markov decision process (POMDP) has been shown to be an effective mathematical framework for dialogue policy learning since it can model uncertainties such as those caused by speech recognition errors, NLU errors, and state tracking errors (Roy, Pineau, and Thrun, 2000; J. D. Williams and S. Young, 2007; S. Young et al., 2013b). In the POMDP framework, the dialogue manager is trained using reinforcement learning (RL) where the agent learns how to act based on the reward signals received from the environment (Sutton and Barto, 1998). The set of actions for each state is domain specific and requires to be defined in advance before training an RL agent. An action is often defined by specific slot-value pairs that can be either *informed* to the user, *requested* or *confirmed* from the user. For instance in restaurant booking a possible action could be to *"inform(name=Seven Days, location=6th King*

street, food-type=Chinese)" or to *"request(name=Seven Days, time=?)"*. Once the set of actions is clearly defined, a Deep Q-Network (DQN; see Section 3.4) can be trained to pick the best one given a dialog state after each human message. Some techniques use only Reinforcement Learning with DQNs (X. Li, Y.-N. Chen, L. Li, and Gao, 2017) but in cases where the action space gets too large, Policy Gradient algorithms are preferred (Su, Budzianowski, et al., 2017). In both cases, a user simulator is mandatory in order to have an environment in which the system can learn based on its experience. This can potentially limit the capabilities of the learning agent and the only way around it is to have an online system that is learning from real users (Su, Gasic, et al., 2016).

Eventually, the dialog manager is evaluated on its system action accuracy, task success rate, or reward depending on the application (Su, Gasic, et al., 2016; X. Li, Y.-N. Chen, L. Li, and Gao, 2017).

4.4 Natural Language Generator

Eventually, once an action has been selected by the Dialog Manager, it is time to return that action to the user in a proper sentence. The goal of the Natural Language Generator (NLG) system is to map semantic information into natural language.

The most naive approach to this problem is to simply define a set of templates or rules (linguistic) methods to map each possible action to natural language. While being simple, error-free and easy to control, it is also time-consuming and not scalable to larger domains. Another method is to use corpus-based statistical methods (Oh and Rudnicky, 2002). Recurrent neural networks (RNN) have been applied to language generation for both social bots and task-orientated dialogue systems (Sordoni et al., 2015; Vinyals and Q. Le, 2015; Wen, Gasic, D. Kim, et al., 2015).

In general, NLG systems are trained to maximize the conditional log-likelihood of the correct sentence given the dialog action returned by the dialog manager (Wen,

Gasic, D. Kim, et al., 2015). However, it is often the case that the model is repeating itself, thus extra care must be taken to have proper sentences at the output. One can use post-processing rules (Oh and Rudnicky, 2002), or gating and attention mechanisms (Wen, Gasic, Mrksic, et al., 2015). More recently other approaches have been tried such as using syntax dependency trees for generating sentences (Dušek and Jurčíček, 2016) or Variational Auto-Encoders and Generative Adversarial networks to produce more realistic sentences (Hu et al., 2017).

Eventually, Natural Language Generator models are evaluated on overlap metrics such as *BLEU* (Papineni et al., 2002), *METEOR* (Banerjee and Lavie, 2005) and *ROUGE* (C.-Y. Lin, 2004), but human evaluations are still required, as those automatic measures correlate poorly with human judgment (C.-W. Liu et al., 2016).

As we saw, modular dialog systems offer a structured pipeline of different modules to provide an adequate response to the user. However, modular dialogue systems are restricted to task-specific domains, and often require significant human feature engineering, including pre-defining entities the NLU module should detect, slot-value pairs that define a dialog state and the action space of the dialog manager. Although this can work well for narrow domains, it does not necessarily generalize to general purpose dialogue systems. On the other hand, in what is called 'end-to-end' models, we do not require a pre-defined state or action space representation; instead, these representations are learned directly from conversational data. Once an end-to-end model architecture is specified, all that is needed to have the system learn to converse about another domain is to provide new training data for that domain. As the amount of available dialogue data grows and more general-purpose conversational systems are desired, we believe that training end-to-end models without hand-crafted features will yield better performance. In the next chapter we describe our novel approach to build a dialog system for the *ConvAI* challenge.

5

Generation of Candidate Responses

– Il faut qu'on pense, et pour penser il faut parler, on ne pense pas autrement

– We must think, and for thoughts we need words, there's no other way to think

Brice Parain, in the movie 'Vivre sa vie', 1962, by *Jean-Luc Godard*

In the previous chapter we described the main components of conventional dialog systems. In this chapter we present our solution to the *ConvAI* challenge by building a more flexible system. Since the topic of each conversation in the challenge is different we could not rely on hand-engineered dialog states, entities, and actions. Instead our system learns to automatically recognize the topic of the conversation and hence the appropriate information to talk about. On the other hand, fully end-to-end systems are flexible and do not rely on any human features, but also tend to produce generic and short responses (Serban, Sordoni, et al., 2016). Hence we decided to use a multi-step approach like modular systems described above, yet flexible enough for our task. The high-level view of our solution is made of three components: response generation, response scoring and response selection. A description of this procedure can be seen in Figure 5.1 We describe the generation procedure in this chapter and tackle the other two in the next chapter.

The first element of our system is the most crucial one: response generation. In this step we generate multiple candidate responses for a given context (composed of the random news article and the conversation history). At the final step, the system will

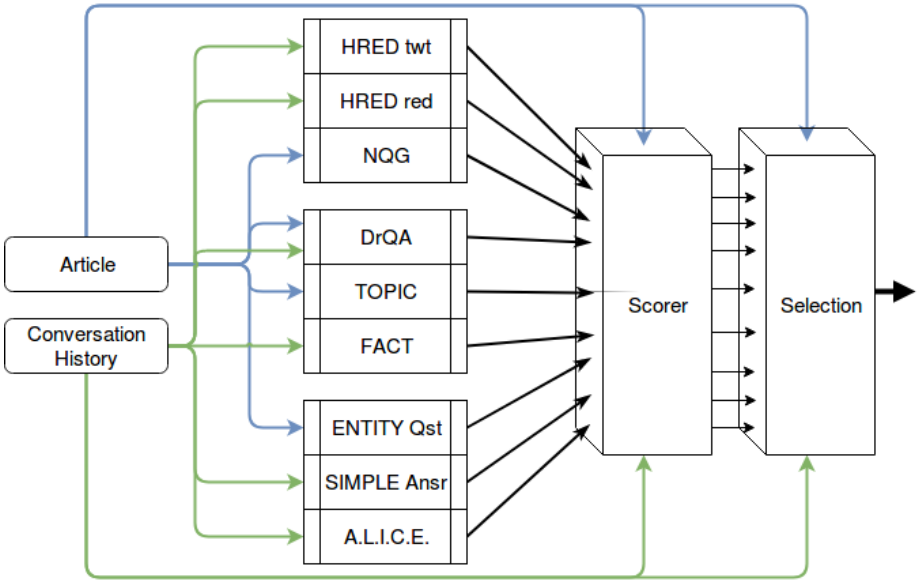


Figure 5.1: High-level view of our chatbot for the ConvAI competition. We follow a 3 step procedure: generation of candidate responses, scoring of all candidates, and selection of one of them.

return one of these candidate responses. It is thus important to produce various type of responses, each of good quality. To that end we decided to use generative sequence-to-sequence models, retrieval-based systems, and rule-based systems. With this wide spectrum of responses we hope to find appropriate answers to all conversations.

5.1 Fully Generative systems

Let us start by describing the different sequence-to-sequence models we used. These are fully generative system, meaning they are generating sentences word by word. The unique challenge of such system is that they need to learn proper syntax and grammatical rules in addition to knowing what to say. To do so, they are only trained on {input - output} sentence pairs. In our chatbot we implemented two distinct models under this scope: the *Hierarchical Recurrent Encoder Decoder* (*HRED* - Serban, Sordoni, et al., 2016) and the *Neural Question Generator* (*NQG* - Du, Shao, and Cardie, 2017).

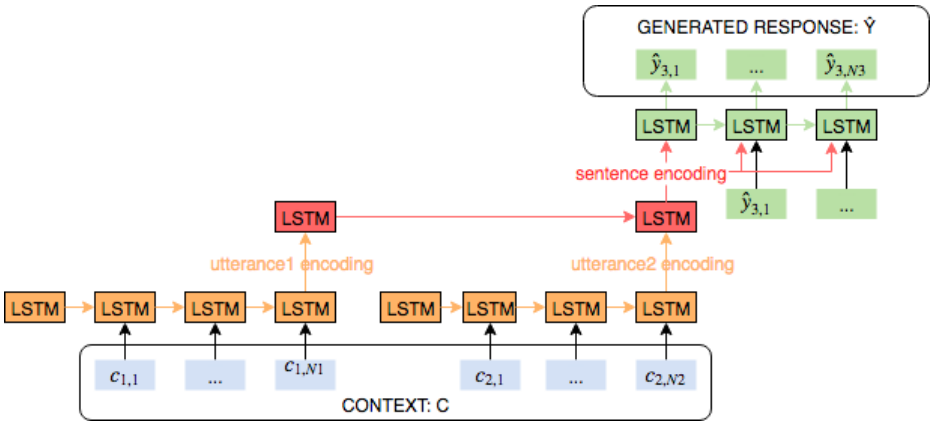


Figure 5.2: The Hierarchical Recurrent Encoder Decoder model with three LSTM networks: one encoder at the word level, a second encoder at the utterance level, and a decoder predicting words of the next utterance. Here the model is given two utterances (c_1 and c_2) in the context and is predicting the third utterance ($\hat{y}_3$) word by word.

5.1.1 Hierarchical Recurrent Encoder Decoder

The *Hierarchical Recurrent Encoder Decoder* (*HRED*) model was initially introduced to build open domain, conversational dialogue systems based on large dialogue corpora using generative models (Sordoni et al., 2015; Serban, Sordoni, et al., 2016). The model is following an encoder-decoder architecture (Cho et al., 2014b) where two (hierarchical) recurrent neural networks *encode* the input sentences into a high-dimensional vector, and a third recurrent network *decodes* that vector representation to output a sequence of words. For all three recurrent networks, the LSTM unit was used (see Section 3.2.2). The first RNN encodes each dialog utterance as one unique vector by having as input word vectors at each time step; the second RNN encodes the entire conversation history as one unique vector by having as input utterance vectors at each time step; and the third RNN decodes, word by word, the next dialog utterance by having as input the previously predicted word and the unique encoded context vector. A pictorial description of this process can be seen in Figure 5.2.

Such model requires a large amount of data to be able to start conversing (Ritter, Cherry, and W. B. Dolan, 2011; Serban, Sordoni, et al., 2016). Thus, we had to

pre-trained this architecture with large and publicly available conversation datasets. We trained two versions of this architecture: one was trained on social interactions, the goal was to have a social component to our chatbot; and another was trained on opinion debates, in that case the goal was to give a personality to our chatbot. The first dataset is made of roughly 1 million conversations scraped from *Twitter* (Ritter, Cherry, and B. Dolan, 2010; Danescu-Niculescu-Mizil, Gamon, and Dumais, 2011); while the other is made of roughly 4 million conversations scraped from *Reddit Politics*, *Reddit News*, and *Reddit Movies*. Each conversation was made of 3 to 6 utterances between two users. Given a context, each model was trained to minimize the negative log likelihood of the next utterance. This is done by projecting the decoded vector of each time step to a vector of vocabulary size, and applying a softmax function on that vector to get the conditional probability of outputting each token in the vocabulary at that time step. More formally we have that the output vector of the decoder at time-step t is:

$$out_t = P(w_t|\hat{y}_1, \ldots, \hat{y}_{t-1}) \in \mathbb{R}^{vocab},$$

where $\hat{y}_{1,\ldots,t-1}$ are the tokens selected in the previous time-steps.

After a week of training, our models were able to have short and generic social interactions such as greetings, and pretending to do some activity. Overall, we found that generated responses from these two models tend to be quite generic and often off-topic. We also note that responses generated by the model trained on *Reddit* data tend to be shorter. Some examples can be seen in Table 5.1.

5.1.2 Neural Question Generator

The second fully generative, sequence-to-sequence model we used in our chatbot was the *Neural Question Generator* (*NQG*). It was originally trained to solve the inverse of the reading comprehension task, that is, instead of answering questions about a piece of text, we automatically generate questions regarding that piece of text (Du,

Context	HRED (twitter)	HRED (reddit)
I met a Tibetan once, he was nice.		
Hello. I don't think the article is about that. What is is about?	I think it's a good thing. I'm not sure. I think I'm a bit of a child.	I'm a man.
Society & Culture		
More or less. It talks about dog!	*laughs* I'm not sure if I'm a bit of a freak. But I'm sure he'll be fine.	I'm a little disappointed that this is a joke.

Table 5.1: Examples of *HRED* responses for specific conversation histories. In all cases the generated responses are off-topic. In addition, responses coming from the model trained on Reddit tend to be shorter.

Shao, and Cardie, 2017). This model was exactly what we wanted as it allowed the chatbot to ask questions about the news paragraph given at each beginning of conversation. This system was trained on the *Stanford Question Answering Dataset* (*SQuAD* - Rajpurkar et al., 2016). Originally, this dataset provides paragraphs of Wikipedia articles with many questions on each paragraph, and the task is to retrieve the span of the paragraph that answers a given question. In order to form a dataset for the question generation task, we retrieved the entire sentence (not just a span of it) that provides the answer to each question for an article. Thus we had {sentence - question} pairs to train the model.

Similarly to the previous generative model, this model is a sequence-to-sequence recurrent neural network following the encoder-decoder architecture (Cho et al., 2014b). One LSTM network encodes the input sentence into a vector representation and the other LSTM network decodes that vector into a question by sampling one word at a time. More specifically, this architecture used bi-directional recurrent neural networks where the input sentence is encoded both from left to right and from right to left. The two vector representations are then concatenated to form one large vector representation. Eventually, an attention mechanism (Bahdanau, Cho, and Bengio, 2014) was also used between the encoder and the decoder LSTMs. This technique allows the decoder network to focus on specific region of the input sentence while generating the

Sentence	Generated Question
The median longevity of mixed-breed dogs, taken as an average of all sizes, is one or more years longer than that of purebred dogs when all breeds are averaged.	What is the average estimate of all dogs?
On 5 December 2011, Pusuke, the world's oldest living dog recognized by Guinness Book of World Records, died aged 26 years and 9 months.	Who recognized the world's oldest living dog?
The dog widely reported to be the longest-lived is "Bluey", who died in 1939 and was claimed to be 29.5 years old at the time of his death.	how many years. longest-lived longest-lived and ?

Table 5.2: Examples of generated questions by the *NQG* model for specific article sentences. The top example is a case in which the answer to the question is not in the source sentence. The middle example is a classic, easy-to-answer question. The bottom example is a case in which the model failed to generate a syntactically correct question.

words that makeup the question, and became a popular feature that is often added to encoder-decoder models. Eventually, like in the previous model, the *NQG* system was trained to minimize the negative log likelihood of the generated question, conditioned on the input sentence.

At the beginning of each conversation we run this model only once on each sentence from the incoming article, and store for later the generated questions. Thus, at any time in the conversation we can use one of these questions on the user without any latency. Overall this system was quite good at generating meaningful questions as we can see in Table 5.2. We also note that sometimes the system generated questions which did not have an answer in the article. This was a nice feature that forced users to attentively read the article and eventually look for that information online if they wanted to have the correct answer. This can be observed in the top example of Table 5.2. The objective of this model was to ask question related to the article and engage the user to read the article.

5.2 Retrieval-based systems

The second category of systems we used in our ensemble of response generation models are retrieval-based. Unlike generative-based systems, those models do not have to learn the syntactic structure of a sentence, instead, their objective is to retrieve the most relevant item (sentence or span or topic) for some specific input, thus making the task much easier. In this category we considered three distinct models: the *Document reader Question-Answering* model (*DrQA* – D. Chen et al., 2017), a topic classifier, and a fact retriever.

5.2.1 DrQA

The *Document reader Question Answering* model (*DrQA* – D. Chen et al., 2017) was introduced to answer open-domain questions using Wikipedia as unique knowledge source. This model was attractive to us as we also wanted to answer as many user questions as possible. However, since the competition forbid us to connect to external sources of information other than the given article, we trained the same model on *SQuAD* dataset (Rajpurkar et al., 2016). As previously mentioned, this dataset provides paragraphs of Wikipedia articles with many questions on each paragraph, and the task is to retrieve the span that answers a given question. That is, given an article and a question, the model is trained to predict the starting and ending position in the paragraph that answers the given question.

The probability of each word in the input paragraph to be the starting or the ending token is computed like so:

$$P_{start}(i) \propto exp(p_i W_s q), \tag{5.1}$$

$$P_{end}(i) \propto exp(p_i W_e q), \tag{5.2}$$

where p_i is a vector representation of token i in the paragraph, W_s and W_e are matrices of learned parameters for the starting and ending probabilities respectively, and q is

a vector representation of the question. The question vector q is a weighted sum of all the hidden units of a bi-directional LSTM network over the word vectors of the question; also known as self-attention technique (Bahdanau, Cho, and Bengio, 2014; Vaswani et al., 2017). The paragraph token representation p_i is the output of a hierarchical bi-directional LSTM network over word vectors at time-step i. This can be seen as the concatenation of the representations of the left and right side of token i in the paragraph.

Overall, we found from our data collection phase[1] that this model was doing a good job at finding the appropriate answers to questions about the article. After training on the *SQuAD* dataset, this model's validation accuracy is 69%. Some of the times, when the information to retrieve is ambiguous or is simply not provided in the article, it can fail. For instance if a user asks for a date and the article is filled with many dates that are all related, it can return the incorrect date. In general however, the model was often correct as we can see in the examples from Table 5.3. The objective of this system was primarily to correctly answer questions, but also to relaunch the conversation by re-stating the article.

5.2.2 Topic Classifier

Another retrieval model we implemented is a simple topic classifier. This model was essential for our application as the most commonly asked question was "*what is this article about?*". We thus wanted to extract an overall topic for any input article. To do so, we trained a text classifier using *fastText* (Joulin et al., 2016) on the *Yahoo News Corpus* (Zhang and LeCun, 2015). This dataset is made of $1,460,000$ news article, each labeled with one topic from a list of ten: "*Society & Culture*", "*Science & Mathematics*", "*Health*", "*Education & Reference*", "*Computers & Internet*", "*Sports*", "*Business & Finance*", "*Entertainment & Music*", "*Family & Relationships*", "*Politics & Government*". The nice thing about these topics is that

[1]Described in Chapter 7.

Article	DrQA Conversation
In addition to the above, Greece is also to start oil and gas exploration in other locations in the Ionian Sea, as well as the Libyan Sea, within the Greek exclusive economic zone, south of Crete. The Ministry of the Environment, Energy and Climate Change announced that there was interest from various countries (including Norway and the United States) in exploration, and the first results regarding the amount of oil and gas in these locations were expected in the summer of 2012. In November 2012, a report published by Deutsche Bank estimated the value of natural gas reserves south of Crete at 427 billion euros.	Where is Greece starting oil and gas explorations?
	Ionian Sea
	Where is the Greek exclusive economic zone?
	south of Crete
	Which countries are interested in the exploration?
	Norway and the United States
	When are the first results about gas and oil expected?
	in the summer of 2012
	How much is the estimated value of the gas reserves?
	427 billion euros
	Which agency published the study about the estimated reserve value?
	Deutsche Bank
	You are so smart!!

Table 5.3: Example of conversation between a human user and the *DrQA* model. The human user starts the conversation by asking a question, and *DrQA* answers in the next message. The conversation alternates like this between human and *DrQA* until the end when the user replies "You are so smart!!".

they cover a wide range of cases and are broad enough so that most of the articles in the competition fall under one of them.

The advantage of using *fastText* is that it is a simple and small model that can run quickly, thus minimizing the user wait time for a response. The classification of every article is done by encoding any article with bag of n-grams features, multiplying that representation by learned parameter matrices, and followed by a Softmax operation to get the probability distribution of belonging to each possible topic. A fast and memory efficient mapping of the n-grams is kept by using a hashing trick (Weinberger et al., 2009; Mikolov, Deoras, et al., 2011). The parameters were trained to minimize the negative log-likelihood of the predicted classes:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n)), \tag{5.3}$$

Topic Sentences
<topic>
This article is about <topic>
I think it's about <topic>
It's about <topic>
The article is related to <topic>

Table 5.4: List of possible sentences to return to the user when asking for the topic of the article. At runtime, we randomly pick one sentence and simply replace "<topic>" by the predicted class from *Yahoo News Corpus*.

where N is the number of articles in that batch, y_n is the n^{th} article label, $f(.)$ is the Softmax operation, A and B the weight matrices being learned, and x_n the normalized bag of n-grams features for the n^{th} article. After training on $1,400,000$ examples, the test accuracy on the held-out $60,000$ example is 61%.

We run this model only once at each beginning of conversation and then store the predicted topic for later if the user asks for it. This allowed us to minimize the answer time to the user. Whenever we use that model we simply return to the user a pre-defined sentence with the predicted topic replacing a generic placeholder. Some examples can be seen in Table 5.4. The objective of this model is to answer the most popular question and make the user think our chatbot actually understands the high level topic of the article. Overall we noticed that the predicted topics were relevant to the given news article.

5.2.3 Fact Retriever

Eventually, the last retrieval model is a simple fact retriever. This model retrieves the most relevant fact to the current conversation from a list of about $2,000$ *interesting* and *fun* facts, including facts about animals, geography and history. The list of facts was shared with authorization from the authors of the *MILABOT* (Serban, Sankar, et al., 2017).

To retrieve relevant facts, we encoded all facts by averaging their pre-trained word vectors, and returned the one that was minimizing its cosine distance with the

average word vector of the conversation history. Note that the fact vectors were computed only once before the challenge, and saved as part of the model. This ensured that we optimized the computation time for the model by only computing the average word vector of the conversation history. The model used pre-trained *Word2Vec* word embeddings (Mikolov, K. Chen, et al., 2013; Mikolov, Sutskever, et al., 2013). Formally, we have the following:

$$dist = F.c^T, \tag{5.4}$$

$$fact = \arg\min_f dist[f], \tag{5.5}$$

where F is the matrix of all fact vectors, c is the conversation history vector, and $dist$ is a list of distances for each fact f. Every time this model is used we select the fact that minimizes its cosine distance with the conversation history and that was not selected before in this conversation.

Once a more or less relevant fact was retrieved from our list, we incorporate it in a randomly chosen pre-defined sentence, like we did for the topic classification model. We also defined a set of prefixes in the case where the user asked a question. Some examples can be seen in Table 5.5. Overall the primary goal of this model was to make the conversation interesting for the user and avoid boredom, but also to have a fun '*exit door*' when the chatbot was not sure what to say. This model was appreciated a lot by the organizers of the competition who found the idea original and effective to keep the conversation fun and interesting.

5.3 Rule-based systems

Finally, the last category of systems our chatbot uses are rule-based models. Unlike the previous two categories of models (except for the fact retriever), these models do not require any training. They are simple yet effective systems that work for specific

Fact Sentences	Prefixes	Some examples of Fact
<fact>	I'm not sure. However, <fact_sentence>	Butterflies cannot fly if their body temperature is less than 86 degrees.
Did you know that <fact>	I'm not sure. But <fact_sentence>	Neurons multiply at a rate 250,000 neurons per minute during pregnancy.
Do you know that <fact>	I'm not quite sure. But <fact_sentence>	The human brain is about 75% water.
Here's an interesting fact, <fact>	I don't have an answer for that. But <fact_sentence>	Flies jump backwards during takeoff.
Here's a fact, <fact>	I don't know. But <fact_sentence>	In every episode of Seinfeld there is a Superman somewhere.

Table 5.5: From left to right: examples of sentences to include a fact, prefixes to use when the user asked a question, and some facts. At runtime, we randomly pick one sentence (and one random prefix if needed) and replace "<fact>" by the most related fact.

cases. We have three such models in our ensemble: the *Entity Sentences*, the *Simple Answers*, and last but not least, *A.LI.C.E. bot*.

5.3.1 Entity Sentences

The first rule-based model we added to our chat bot is a simple model that returns sentences related to entities from the article. We manually defined a set of 50 different sentences and questions with special *entity tags* in them. Once a new article comes in at each beginning of conversation, we parse it using *Spacy Named Entity Recognizer*[2] to recognize the following 10 entities: "*persons*", "*organizations*", "*geographical entities*", "*locations*", "*products*", "*events*", "*work of art*" (books, songs), "*languages*", "*dates*", "*nationalities, religious or political groups*".

After we recognized all entities in the article, we simply replace all appropriate tags by the actual entity found in the article and return a random sentence from our list that was not used before. Some example of sentences can be found in Table 5.6. Overall the objective of this model is to increase the user engagement by talking about things related to the article and by asking simple questions.

[2] https://spacy.io/api/annotation

Examples of Entity Sentences
Do you know what <person> did in his life ?
Have you ever used any of <orgs>'s product or services ?
What do we eat in <gpe>? I'm starving!
Have you ever been to <loc> ? I heard it's beautiful.
Once, I bought a <product>, but then somebody stole it from me.
What do you think about <event> ?
Do you know who did <work of art> ?
Do you know how to speak <language> ?
What happened in <date> ?
I met a <norp> once, she was nice.

Table 5.6: Examples of entity sentences for different named entity types. In order from top to bottom: "<person>" (people, including fictional); "<orgs>" (companies, agencies, institutions, etc.); "<gpe>" (countries, cities, states); "<loc>" (mountain ranges, bodies of water); "<product>" (objects, vehicles, foods, etc.); "<event>" (named hurricanes, battles, wars, sports events, etc.); "<work of art>" (titles of books, songs, etc.); "<language>", "<date>", "<norp>" (nationalities or religious or political groups).

Trigger sentences	Pre-defined answers
How are you ?	I am great! What about you?
What are you ?	I am a chatbot.
Who made you ?	I am a chatbot developed by students at McGill University.
What's your name ?	My name is RLLChatBot.
Where do you live ?	I can live everywhere at anytime.

Table 5.7: Examples of sentences captured by our regular expressions and the possible sentence that the model could return.

5.3.2 Simple Answers

Another rule-based model we added to our chat bot is a simple regular expression catcher. We defined a set of regular expressions for generic and unrelated questions that the user can ask, and a list of appropriate answers to return for each case. Unlike all the other models previously described, this one is only returning a response when the previous user message contains an expression that matches our regular expression parse. This model was primarily used to make a personality for the chatbot, handle unrelated questions, and edge cases. Some examples can be seen in Table 5.7.

Context	A.L.I.C.E.
Hello chatbot! What is this article about?	Hi there! I am a social robot from earth. Ask me anything about news, politics or movies!
It's about Science & Mathematics	
Ok, can you tell me anything more about it?	Excuse me! What I want to tell you is, how much fun it is talking with you.
Excuse me! What I want to tell you is, how much fun it is talking with you.	
lol ok not much apparently. What happened in 1906?	I'm glad you find this amusing. Aha. That is a non-committal reply. I don't know what happened.
I don't know. But here's a fact! <fact>	
hmmm ok good to know	Pardon me? are you shy? Glad to hear it.

Table 5.8: Examples of *ALICE* responses for specific conversation histories.

5.3.3 A.L.I.C.E. Bot

The final model that produces a candidate response is the *Artificial Linguistic Internet Computer Entity* (*A.L.I.C.E.* – Wallace, 2009)[3]. This chatbot is a rule-based system based on *Artificial Intelligence Markup Language* (*AIML*). Released in 1995, it is the first *AIML*-based personality program who won the *Loebner Prize* as "the most human computer" at the annual Turing Test contests in 2000, 2001, and 2004. It is composed of roughly 20,000 conversation rules with patterns to recognize and template answers to return.

We run this chatbot after every user message to produce a possible response. Since the response only depends on the conversation history and not the article, this model was mostly used to maintain a social-ish and coherent conversation with the user. We found that this model was important to our pool of models as it was giving a fake personality to our chatbot. Moreover, we found that *ALICE* covers a wide spectrum of edge cases and the regular expressions we manually crafted for the previously described model were not that useful after adding *ALICE*. Some example of conversation can be seen in Table 5.8

[3] https://sourceforge.net/projects/alicebot/

6

Scoring and Selection of Responses

> – Le plus difficile dans l'art du dialogue, ce n'est pas de parler, c'est d'apprendre à écouter.
> – The most difficult thing in the art of dialogue is not to speak, it is to learn to listen.
>
> *Jean-Marie Petitclerc*

In the previous chapter we presented the ensemble of models that produce a possible response for a given article and a conversation history. In this chapter we describe our mechanism to decide which response to return to the user. This process is done in two steps: we first score each candidate response, and then select one response based on its score and the conversation state.

6.1 Scoring of candidate responses

After generating several candidate responses in parallel, the chatbot has now the obligation to make the hard choice of picking only one response for the user. To help in this decision, we give a score to each possible response. We now present two different approaches to do so: the first is based on classification with supervised learning, while the other is based on reinforcement learning.

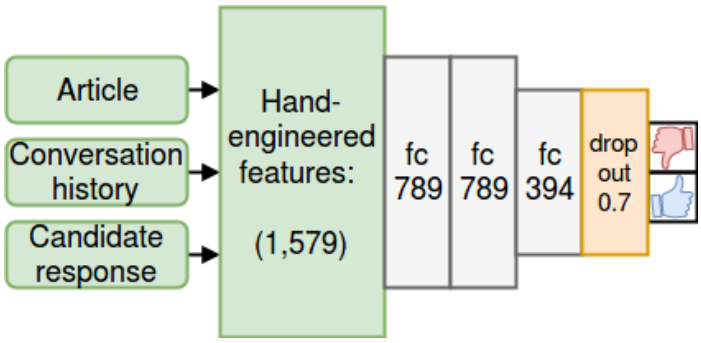


Figure 6.1: Architecture of the feed-forward network to predict candidate responses' vote. We use 3 fully connected feed-forward layers of dimension 789, 789, and 394. The input is of dimension 1579 and the output of dimension 2.

6.1.1 Supervised Scoring

Our first attempt at scoring the candidate responses produced by our ensemble of models relies on *supervised learning*. Remember that in the competition framework the human participant could up-vote or down-vote the response from the other participant in a conversation. This vote gave us some important feedback on the response quality being presented to the user. We thus decided to build a classifier that could predict the human vote for a given response, conditioned on the conversation history and the article.

We used the competition data to make a dataset of this form: $\{x_i, y_i\}_{i=1}^{n}$ where x_i is a vector representation of the article, the conversation history (or the *context*), and the candidate response; and $y_i = 1$ for up-voted responses and 0 otherwise. We then fed this input vector x_i to a fully connected feed-forward neural network with a softmax layer at the output in order to have the probability of the input being an up-voted response. We also added a dropout layer (Srivastava et al., 2014) before the last layer to prevent our network from overfitting on our *small* training data of $7,119$ instances. The architecture of the network that we used can be seen in Figure 6.1. We trained all parameters of the network to minimize the cross-entropy loss function (see Equation 3.25) between the predicted probabilities and the true vote of each message. The training algorithm is described in pseudo code by Algorithm 1.

In order to represent the input information as a vector, we manually created a set of features that we considered important for making that prediction. This allowed us to train fewer parameters and work with the small dataset that we had at the time of the competition, composed of 1,750 unique articles and 8,902 instances of {article, context, message, vote}. We list below the set of features we manually computed for each (article, context, candidate) triples:

- Average word embeddings of the candidate response, the previous messages (context), and the article. We used pre-trained word2vec embeddings (Mikolov, K. Chen, et al., 2013) for each of them.

- Similarity metrics between the candidate response & the context, and between the candidate response & the article. We used average embedding cosine similarity, extrema embedding score (Forgues et al., 2014), and greedy matching score (Rus and Lintean, 2012) with pre-trained word2vec embeddings (Mikolov, K. Chen, et al., 2013).

- Number of non stop-words, bi-grams, tri-grams, and spacy entities overlap between the candidate response & the context, and between the candidate response & the article.

- If the candidate response is generic. We defined a message to be generic if it is made of stop-words or words shorter than 3 characters. We did the same for the last user message in the context.

- If the candidate response has: one or more words starting by 'wh', one or more intensifier words (e.g. *amazingly*, *crazy*, and so on), one or more confusion words (e.g. *confused*, *stupid*, *nonsense*, and so on), one or more profanity words, and one or more negation words (*not* or *n't*). We indicate the presence of each of these categories by a 1 and the absence by a 0. We did the same for the last user message in the context.

- The number of messages in the conversation so far.
- The number of sentences in the article.
- The number of words in the candidate message and in the last user message in the context.
- The type of the candidate response. We define a type to be any combination of '*greeting*', '*question*', '*affirmative*', '*negative*', '*request*', or '*politic*' message. We perform a heuristic decision based on word presence for each of the types. We did the same for the last user message in the context.
- Sentiment score (negative, neutral, or positive) of the candidate response. We used the pre-trained Vader sentiment analyzer (Gilbert, 2014). We did the same for the last user message in the context.

The combination of all these features made a vector of $1,579$ values that we use as the input x_i to our classifier.

For our system submitted to the *ConvAI* challenge, we searched the best parameter combination by running 100 experiments (in a random parameter search fashion) and evaluated the classification accuracy on a validation set with early stopping and a patience of 20 epochs. In the end, the model with highest validation accuracy was trained with a batch size of 128, the *RMSProp* optimizer (G. Hinton, Srivastava, and Swersky, 2012), a learning rate of 0.001, *ReLU* activation functions, and a dropout rate of 0.70. This combination of parameters gave us a validation accuracy of 64.23% and a training accuracy of 94.77%. Eventually, at runtime, for each candidate response, we can compute its corresponding feature vector x_i and pass it to our neural network, the output gives us a vector of probabilities $\hat{y}_i$ representing the probability of the candidate response being up-voted or down-voted. In order to give a score to the candidate response we simply pick its probability of being up-voted.

Algorithm 1: Classifier Training Algorithm

inputs: D^{train}: a list of input-output pairs
D^{valid}: a list of input-output pairs
θ: weights for the classifier network
$maxepoch$: number of maximal training epochs (10,000)
$patience$: patience term (20)

$best \leftarrow 0$;
$pt \leftarrow patience$;
while $pt > 0$ *and epoch* $e \in \{1, 2, ..., maxepoch\}$ **do**
  foreach *batch* B *of* 128 *examples* $\in D^{train}$ **do**
    $pred^{train} \leftarrow MLP(B|\theta)$;
    $loss^{train} \leftarrow crossentropy(pred^{train}, B)$;
    $\theta \leftarrow RMSProp(\theta, loss^{train}, 0.001)$;
  end
  $pred^{valid} \leftarrow MLP(D^{valid}|\theta)$;
  $acc \leftarrow accuracy(pred^{valid}, D^{valid})$;
  if $acc > best$ **then**
    $best \leftarrow acc$;
    Save θ;
    $pt \leftarrow patience$;
  else
    $pt \leftarrow pt - 1$;
  end
end

6.1.2 Q-Scoring

The other scoring mechanism we implemented after the *ConvAI* competition relies on *reinforcement learning*. Instead of predicting the immediate reward of a candidate response (the up- or down-vote), we try to estimate the Q-value of that response. Recall from Section 3.4 that this Q-value is an estimate of the expected future reward after taking that action (i.e.: selecting that response). In this setting we consider the *state* of the environment as the article and the conversation history, the chatbot's *action* is defined to be the candidate response to return to the user, and the *reward* of taking such action is a weighted version of the up- or down-vote signal. When the response was up-voted, the reward was defined to be 0.2 if the end-of-conversation score was 1 ('very bad') or 2 ('bad'), 0.8 if the end-of-conversation score was 3 ('medium') or

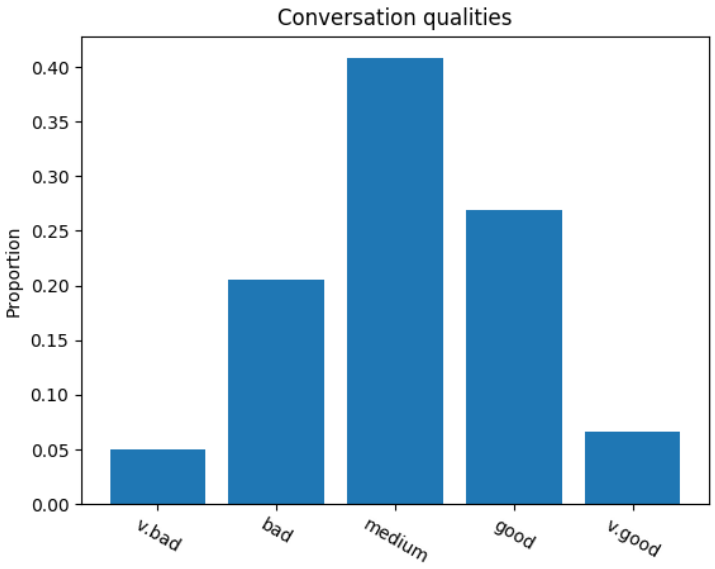


Figure 6.2: Proportion of end-of-conversation score given by human users during our data collection phase.

4 ('good'), and 1.0 if the end-of-conversation score was 5 ('very good'). Given that users tend to be neutral in their evaluation (as we can see in Figure 6.2), we decided to strongly penalize 'very bad' and 'bad' conversations because these had poor meaning, while 'medium', 'good' and 'very good' conversations were coherent. When the response was not up-voted its reward was set to 0.

In order to predict this Q-value, we collected a large amount of data in addition to the competition dataset, and trained a Deep Q-Network (DQN) to imitate the human behavior found in the data. This means that rather than performing classical Q-learning where the agent is interacting with the environment while training, we implement a form of *Neural Fitted Q-Iteration*. We decided to do so primarily for practical reasons. The only way to have our DQN learning while interacting is to have a simulation of a human user that could up-vote or down-vote responses while talking with the chatbot. However, having this simulator is the same as solving the original problem. On the other hand, asking real users to do so would be impractical given the time response of real users, their personal patience to exploratory behaviors

from our chatbot, and their adaptive evaluation as the chatbot would have become better over time. We describe how we collected trajectories of conversations while avoiding all these problems in the next chapter. For now, let us assume that we have a collection of *expert* trajectories that we want to imitate of the form: (s, a, r, s') where s is the current state of the environment (article & conversation history), a is an action (a candidate response), r is the reward of taking that action (between 0 and 1 as described above), and s' is the next state after taking action a (article & new conversation history including action a and the human response to a).

To train our DQN, we could use the simple architecture above that uses hand-engineered features, but there is no real distinction between the *state* (article & conversation) and the *action* (candidate), as they are both combined into the same input vector. Thus, we use recurrent neural networks to automatically represent the state and the action in separate vectors. We also take inspiration from the *dueling* architecture (Wang et al., 2015) that splits the prediction of the Q-value ($Q(s, a)$) as the sum between the state value ($V(s)$) and the advantage function ($A(s, a)$). This has the advantage of predicting a value for the state on its own and for each action separately, empirically yielding better results. We use hierarchical Gated Recurrent Unit (GRU) networks with shared weights to encode the article, the conversation history, and the candidate response into vectors. Similar to the *HRED* generative model, each sentence in the article (and each message in the the conversation) is first encoded into a sentence vector before encoding the list of all sentence vectors as one article (or context) vector. The concatenation of the article and context vectors gives us a vector representation of the *state*. The candidate responses are simply encoded with the first layer GRU, no hierarchy is used. These candidate response vectors give us vector representations of each next possible *action*. We then pass the state and action vectors to fully connected feed-forward networks and compute separately a state value $V(s)$ and an advantage value $Adv(s, a)$. The final Q-value is defined as $Q(s, a) = V(s) + Adv(s, a)$. A visual description of the architecture can

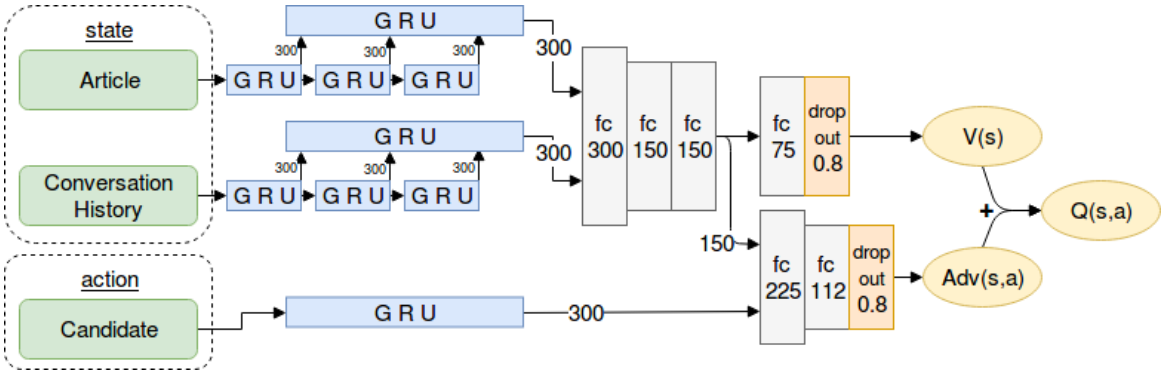


Figure 6.3: Architecture of our Deep Q-Network to predict candidate responses' action value. We use hierarchical GRU networks with shared weights, and different fully connected feed-forward layers. The input word vectors and both GRU's output vectors are of dimension 300. The fully connected layers reduce the dimensionality at each depth and the output is a simple scalar.

be seen in Figure 6.3. The network is trained to minimize the Huber loss function (see Equation 3.27) between the current estimate of the Q-value and the expected Q-value based on the observed reward (also called the *target*). We use the Double DQN target (Van Hasselt, Guez, and Silver, 2016b) in order to have a better estimate, as described by Equation 3.36. The training algorithm is described in Algorithm 2. Eventually, at runtime, for each candidate response, we can compute its corresponding Q-value based on the current state of the environment by feeding its word vectors to our DQN. In order to give a score to the candidate response we simply pick the output of our network. Experiments with this model were done after the competition and are described in the next chapter.

6.2 Selecting one response

After generating several candidate responses in parallel and scoring each of them with one of the mechanism described above, the chatbot has now the obligation to make the hard choice of picking only one response for the user. There again, two approaches are considered: we can either follow a set of rules based on the conversation state,

Algorithm 2: DQN Training Algorithm

inputs: D^{train}: a list of $(s, a, r, s', (a'_i)_{i=1}^n)$ tuples
D^{valid}: a list of $(s, a, r, s', (a'_i)_{i=1}^n)$ tuples
θ: weights for the first DQN network
$\bar{\theta}$: weights for the target DQN network
τ: frequency at which to update target net (2,000)
γ: discount factor (0.99)
maxepisode: number of maximal training episodes (10,000)
patience: patience term (20)

$best \leftarrow +\infty$;
$pt \leftarrow patience$;
while $pt > 0$ *and episode* $e \in \{1, 2, ..., maxepisode\}$ **do**
  foreach *batch* $B = (s, a, r, s', (a'_i)_{i=1}^n)$ *of* 128 *examples* $\in D^{train}$ **do**
    $Q_\theta^{train}(s, a) \leftarrow DQN(s, a|\theta)$;
    $a^{max} \leftarrow \arg\max_{a'_i} Q_\theta(s', a'_i)$ such that $Q_\theta(s', a'_i) = DQN(s', a'_i|\theta)$;
    $Q_{\bar{\theta}}^{train}(s', a^{max}) \leftarrow DQN(s', a^{max}|\bar{\theta})$;
    $target^{train} \leftarrow r + \gamma Q_{\bar{\theta}}^{train}(s', a^{max})$;
    $loss^{train} \leftarrow huber(Q_\theta^{train}(s, a), target^{train})$;
    $\theta \leftarrow ADAM(\theta, loss^{train}, 0.0001)$;
    if $step \mod \tau = 0$ **then** $\bar{\theta} \leftarrow \theta$;
  end
  $(s, a, r, s', (a'_i)_{i=1}^n) \leftarrow D^{valid}$;
  $Q_\theta^{valid}(s, a) \leftarrow DQN(s, a|\theta)$;
  $a^{max} \leftarrow \arg\max_{a'_i} Q_\theta(s', a'_i)$ such that $Q_\theta(s', a'_i) = DQN(s', a'_i|\theta)$;
  $Q_{\bar{\theta}}^{valid}(s', a^{max}) \leftarrow DQN(s', a^{max}|\bar{\theta})$;
  $target^{valid} \leftarrow r + \gamma Q_{\bar{\theta}}^{valid}(s', a^{max})$;
  $loss^{valid} \leftarrow huber(Q_\theta^{valid}(s, a), target^{valid})$;
  if $loss^{valid} < best$ **then**
    $best \leftarrow loss^{valid}$;
    Save θ;
    $pt \leftarrow patience$;
  else
    $pt \leftarrow pt - 1$;
  end
end

or simply return the message that has the highest score according to the scoring mechanism used.

For the competition, we built a set of rules that takes in account the simple classifier (described in Section 6.1.1) to score each candidate response. This was found to work best at the time because the classifier validation accuracy was not good enough (only 64.23%, when a random binary classifier can have 50%) to only rely on it and we did not have enough data to train the deep recurrent networks of our DQN architecture described in Section 6.1.2. Every conversation starts with the following pattern: a random wiki-news article's paragraph is selected and sent to both users, we then present the participant with a pre-defined greeting message ("*Hello! I hope you're doing well. I am doing fantastic today! Let me go through the article real quick and we will start talking about it.*") and a sentence from either the *Neural Question Generator* model (Section 5.1.2) or the *Entity Sentences* model (Section 5.3.1). We randomly pick one of these two models as they are best suited to start a conversation by asking an open question. The user is then free to reply. Based on the user's reply, we generate a set of candidate responses, score each of them, and follow the rules described below in order of specificity (also illustrated in Figure 6.4).

1. Our most specific model with no flexibility at all is the rule-based *Simple Answers* model (Section 5.3.2). That is why we first check if the user's message matches our set of simple regular expressions that try to handle edge-cases and weird messages from the user. If it does, we simply reply with the corresponding pre-defined answer. If it does not, we proceed to the next rule.

2. The next specific item we need to check is if the user asks about the topic of the article since this behavior is to be expected in a challenge where the task is to talk about a given article. If the user's last message match our set of regular expressions designed to catch these messages, we return the response provided by the *Topic Classifier* model (Section 5.2.2). If not, we proceed to the next

rule.

3. Another important and common scenario we expect to happen is when the user asks a question about the article. We want to make sure to catch those cases because our *DrQA* model (Section 5.2.1) is specifically designed to answer questions. As such, if the previous user's message has a common entity with the article, terminates by a question mark, and has a 'wh-'word, we assume that the user asked a question about the article, and we return the response provided by *DrQA*. If the user's response does not match those characteristics, we proceed to the next rule.

4. In order to keep the conversation interesting for the participant, we want to see when the user could be bored and remedy that. We thus introduce a 'bored' counter that we increment every time the user response is short (less than 3 words) or when the user response is made entirely of stop-words. As soon as this counter reaches 2 (i.e. the user sent a 'bored' message twice in the conversation) we sample a response from the *Neural Question Generator* model (Section 5.1.2), or the *Fact Retriever* model (Section 5.2.3), or the *Entity Sentences* model (Section 5.3.1) according to their score (given by our up-vote classifier) and reset the counter to 0. We sample one of these models as they are best to re-launch the conversation, and potentially start talking about something new. If the counter does not reach 2 yet or if the user is not considered 'bored', we proceed to the next rule.

After the previous four rules, we hope to manage all cases in which some specific models are useful for. Thus, we now let our trained scoring mechanism handle the other types of user messages.

5. If a candidate response from one of the *HRED* models (Section 5.1.1) or the *A.L.I.C.E.* model (Section 5.3.3) has a high score (between 0.75 and 1.0), we

return the candidate with the highest score. The intuition here is that if the scoring mechanism is strongly confident, we should use that response. Note that we only consider those three models as they are the most flexible ones and are thus considered the best for generic conversations. If none of these responses have a high score, we continue with the next rule.

6. When the scoring mechanism is not strongly favouring a certain response, we decide to split user messages in two categories: they either ask a question, or they do not. Note that if they ask a question related to the article, our third rule would have returned a *DrQA* response. However, users may also ask unrelated questions. As such, if the user response terminates by a question mark and has a 'wh-'word, we sample a response from either one of the *HRED* models (Section 5.1.1), or the *A.L.I.C.E.* model (Section 5.3.3), or the *DrQA* model (Section 5.2.1) based on their score (given by our up-vote classifier) as long as their score is not too low (greater than 0.25). We consider these models because they are all quite flexible in terms of their type of responses and thus potentially capable of answering a broad range of questions. Note that we do not consider models that could ask a question because at this stage of the conversation we try to answer the user's question. If the user's response does not fall into these characteristics, we follow the next rule.

7. When the user response does not contain a question, we sample a response from either one of the *HRED* models (Section 5.1.1), the *A.L.I.C.E.* model (Section 5.3.3), the *Neural Question Generator* model (Section 5.1.2), or the *Entity Sentences* model (Section 5.3.1) based on their score (given by our up-vote classifier) as long as their score is not too low (greater than 0.25). We use these models because they are all quite flexible in terms of the type of responses and thus great for generic conversation. Note that in this case we do consider models that can also ask questions since we assume that the user did not ask

one in his or her last message.

8. Eventually, if none of the above scenario is available (i.e. most responses have a low score, below 0.25), we simply return a more or less related fact based from the *Fact Retriever* model (Section 5.2.3). We introduce this rule as a safe '*exit door*' for our chatbot so that it always has a response available if there is a problem with other models.

After selecting a response to return to the user, we wait again for the user to reply to our chatbot and select the next response based on the same rules following the same order again, until the participant decides to terminate the conversation.

Eventually, our goal is to remove most of the above rules and let the scoring machine completely decide which response to return to the user entirely based on its score. We can then replace all these rules by either a *hard* decision: $resp = \arg\max_r Score(r|article, context)$ or a *soft* decision by sampling a response rather than always taking the best one: $resp \sim Score(r|article, context)$. This can make our chatbot slightly more flexible and less relying on our human expertise about how conversations are supposed to go. To that end, we collected a larger dataset of conversations and trained both a reward classifier model and a Q-estimator model using both the simple MLP architecture described in Section 6.1.1, and the DQN architecture described in Section 6.1.2. The next chapter presents our data collection procedure and the different experiments we ran trying to improve the current selection process.

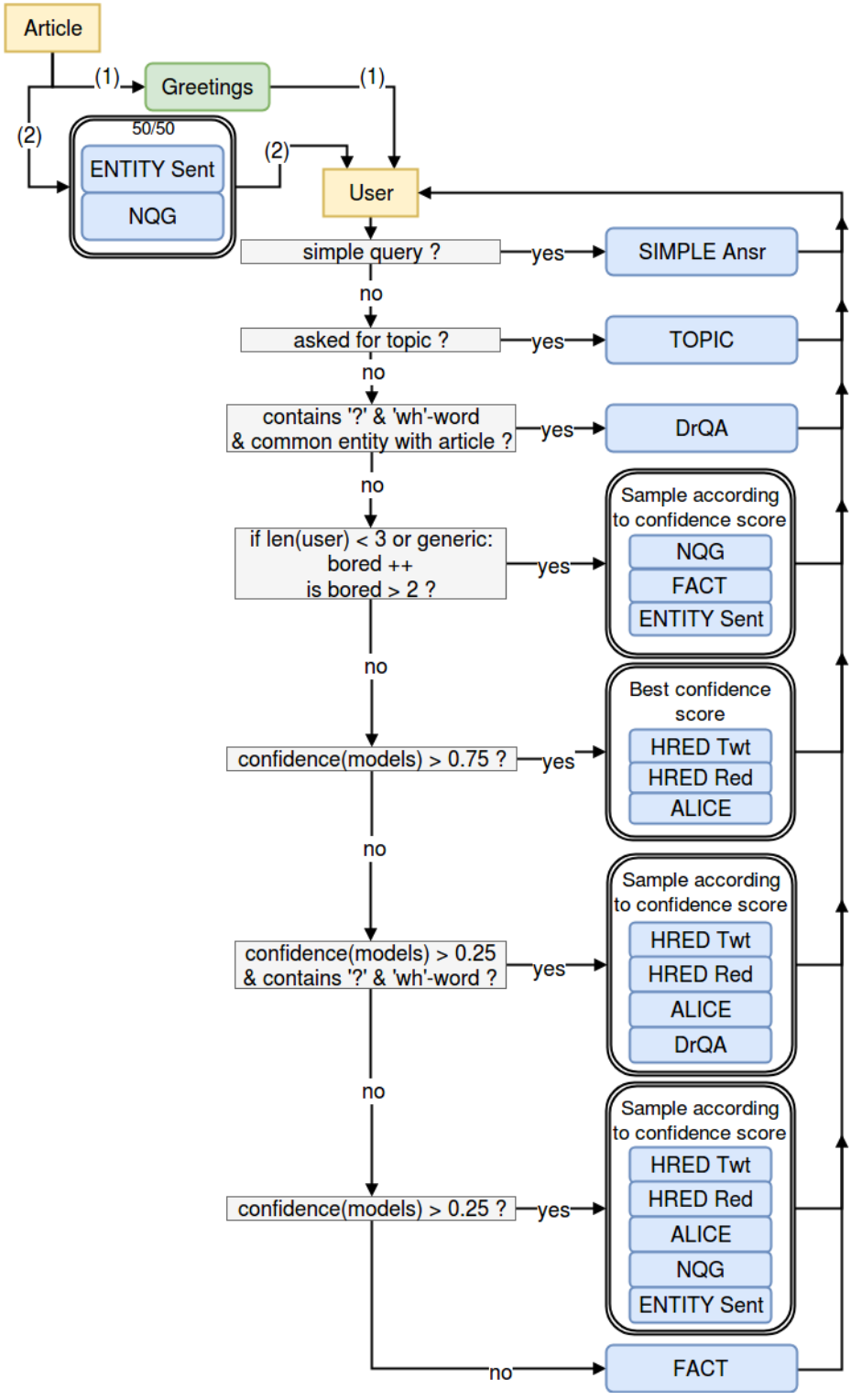


Figure 6.4: Set of rules to follow during a conversation in order to decide which response to return to the user based on both the conversation state and the score of each possible candidate response. The random article first comes in the conversation, we then greet the user and ask a question. After each user's response we then follow these rules.

7

Data Extension and Experiments

> – It's not an experiment if you know it's going to work.
>
> *Jeff Bezos*

In the previous chapter we presented two procedures to score candidate responses and a rule-based selection mechanism. We also mentioned that, at the time of the competition, we only had access to a small amount of conversational data ($1,750$ unique articles and $8,902$ instances of {article, context, message, vote} tuples), which motivated our use of the smaller scoring network (see Section 6.1.1), the use of hand-engineered features, and of rule-based message selection. In this chapter we present additional data independently collected after the *ConvAI* challenge in order to train the more complicated architecture described in Section 6.1.2 that automatically extracts features rather than relying on hand-crafted ones. Furthermore, we present the different experiments we ran with this additional dataset and compare results between the use of rule-based message selection against using the scoring mechanism to decide which response to select and return to the user.

7.1 Data Collection

In order to expend the dataset collected during the *ConvAI* challenge, we use Facebook's *ParlAI* framework[1] to ask workers from *Amazon Mechanical Turk* to chat with our chatbot. In order to follow as much as possible the challenge structure, we start every conversation with a random paragraph from a random *SQuAD* article (Rajpurkar et al., 2016), present the same greeting message we had at the time of the competition, and start by asking a question about the article's paragraph by running both the *Neural Question Generator* model (*NQG* – Section 5.1.2) and the *Entity Sentences* model (Section 5.3.1). The difference with the competition is that in our case, rather than asking the user to up- or down-vote the response our rule-based selection criteria would have chosen, *we let the user decide* which candidate response he or she prefers. After selecting **one** response, the user has to write a reply to this bot utterance he or she previously selected. Every following interaction follows the same pattern: we present a list of candidate responses (all nine models described in Chapter 5 are used, except for the first interaction for which we use only *NQG* and *Entity Sentences* models), the user picks one, write his or her reply, and we present again a list of all possible bot replies. A visual description of the user interface used during the data collection can be seen in Figure 7.1. After a minimum of 5 interactions, the user can decide to finish the conversation, in which case he is asked to provide a score between 1 ('very bad'), 2 ('bad'), 3 ('medium'), 4 ('good') and 5 ('very good') for the entire conversation. We specifically ask users to ignore bot responses that were not selected during the chat when giving this final score. That way, we get a fair evaluation of the actual conversation that the user just had.

It is important to note that the nine different models that generate candidate responses (described in Chapter 5) were built and trained only once before the competition and remained fixed thereafter. This allows us to fix the capacity of our

[1] http://parl.ai/

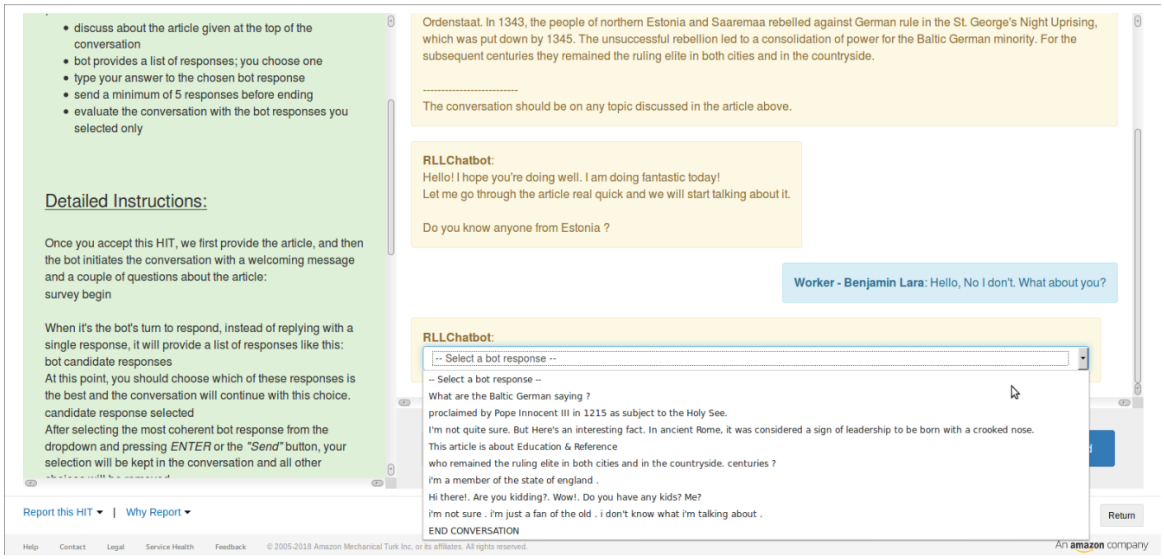


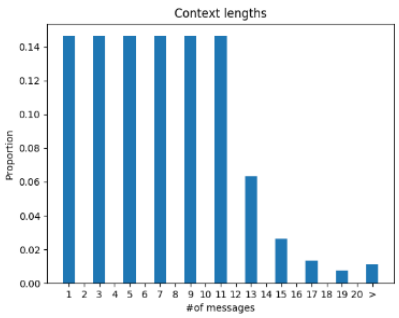
Figure 7.1: User interface of *Amazon Mechanical Turk* during our data collection phase. We first send a random paragraph from a random *SQuAD* article's paragraph and a greeting message. The user then selects the bot response '*Do you know anyone from Estonia?*' and replies. We then again present the user with a choice of potential responses. He has to pick one. We also present detailed instructions on the left side of the screen for the user to refer at any time during the conversation.

chatbot during the entire data collection process and work with a fixed environment in which we can better compare different scoring and selection mechanisms. In addition, if a model has many possible candidate responses for each time step in the conversation (such as the *NQG* model, cf: Section 5.1.2), we simply pick one random candidate response that was not presented to the user before in this conversation. If all possible candidate responses for the same model have been shown, we simply select a random response from that model.

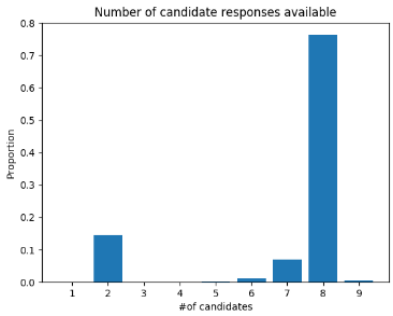
We let the user decide which response our bot should return rather than exploring different selection behaviors ourselves since we are less prone to bore the human user with bad policies and it is much more data efficient. Indeed, for each interaction, we now have both selected and non-selected responses. On the other hand, during the *ConvAI* challenge, we only had 1 response per interaction with its corresponding vote. Here, we consider the selected response as *up-voted*, and all the non-selected responses as *down-voted*.

#of conversations	1,765
#of unique articles	1,663
#of interactions	12,057
#of transition tuples	70,761
Avg. #of available candidate responses per *interaction*	7.02165
Avg. #of messages per *context*	7.12491

(a)



(b)



(c)

Figure 7.2: Statistics on the entire additional data collected with *Amazon Mechanical Turk*. (a) Table of different data quantities. (b) Proportion of the number of messages per context. (c) Proportion of the number of available candidate responses per interaction.

The statistics of the data we collected can be seen in Figure 7.2. We define one *interaction* as one bot response followed by a human response, and a *transition tuple* as one data instance of this form: {article, context, candidate response, score} with the *score* being 1 or 0 depending if the candidate was selected or not by the human user. We can see from histogram (b) that there is an equal number of data instances with context length being 1, 3, 5, 7, 9, and 11. That is to be expected since human users were asked to perform a minimum of 5 interactions before ending the conversation. We also see that a few users continued their conversation further, which is a good sign for our chatbot because it probably means that they enjoyed chatting with the bot since we only paid them for 5 interactions. The average number of interactions per chat is ~7 from table (a). Moreover, we note that there is only an odd number of messages in every context simply because we consider the first greeting message from our bot as the first context message, then the user has to pick a candidate (considered as the second context message), and reply to it (considered as the third context message). Thereafter we consider one interaction as 1 bot message selection followed by one user reply, as mentioned above. When we allow the user to terminate the conversation, there are 11 messages in the context and the user can decide to either pick another candidate response to continue the chat or finish the conversation

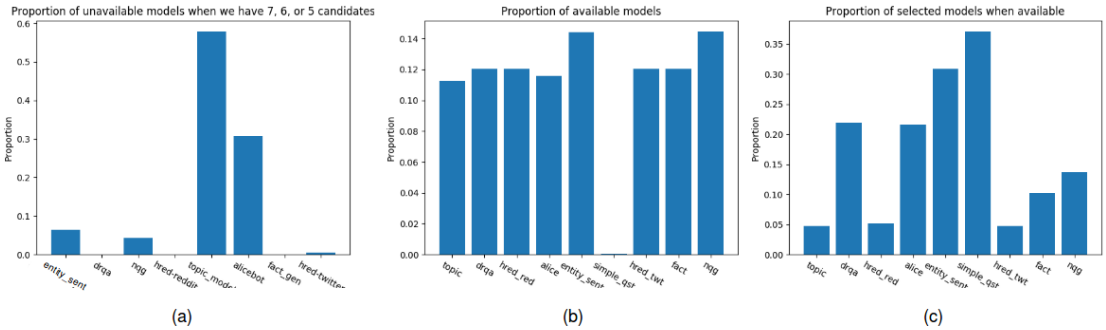


Figure 7.3: Statistics on the different candidate model usage. (a) Proportion of the number of times each model (except the *Simple Answers* model) was *not* available when the number of candidate responses was 5, 6, and 7. (b) Proportion of the number of times each model was in the list of available candidate responses. (c) Proportion of the number of times each model was selected *when it was in the list of available candidate responses*.

now and give it a final score.

From histogram (c), we can see that most of the time (76.32% of our data instances) the number of candidate responses available to the user is 8. That is to be expected since from the nine generative models that we have, the *Simple Answers* model (Section 5.3.2) provides a response only when the previous user message in the context matches our small set of regular expressions (shown in the diagram by the small number of times the number of available candidate responses is 9 – 0.51% instances). All the other models are expected to return a response at all times. We also note that quite a few times (14.64% instances), only two candidate responses are shown to the user, which is also to be expected: indeed, at every beginning of conversation we only trigger the *Neural Question Generator* model (Section 5.1.2) and the *Entity Sentences* model (Section 5.3.1) as previously explained. However, we can also notice that some rare times the number of candidate responses available to the user is only 7, 6, or even 5 (8.48% of our data instances in total). This is due to some models not providing responses.

To understand a little more why this happens we present in diagram (a) of Figure 7.3 some statistics about which model is not available to the user when there

are only 5, 6, or 7 candidate responses, ignoring the *Simple Answers* model (Section 5.3.2) that is only available in rare cases. We can see that the *Topic Classifier* model (Section 5.2.2) and the *ALICE* model (Section 5.3.3) are mostly responsible for cases in which the user is missing some candidate responses (~88.62% of the time). After some investigation, we found that the topic *Classifier* we used (*fastText* – Joulin et al., 2016) is running in a separate thread from the actual *Topic Classifier* model thread. Recall that we also have one thread for each other generative models. Thus, with all the other threads already running, it is possible that the *fastText* classifier thread takes time to start. If started before the second interaction of the conversation (first time we request the *Topic Classifier* model), the *fastText* classifier may not always have time to classify the article's paragraph before that. In both cases, if the *fastText* classifier does not have a label for the article's paragraph before the second interaction of the conversation, the *Topic Classifier* model is unable to return a candidate reply. The *ALICE* model is the second model that most often fails to return a response. After investigating the original code we downloaded for *ALICE*, we noticed that when the response from *ALICE* contains quotation marks, the bot assumes that the response contains profanity language and returns an empty response only. We show some failure examples in Table 7.1. Other models that rarely fail to return an answer are the *Entity Sentences* (Section 5.3.1), and the *NQG* (Section 5.1.2) models. The most common reason for failure in those cases is when the article's paragraph is only one sentence. Indeed, in short paragraphs there is not always enough information for our models to return a response.

Furthermore, we show in diagram (b) of Figure 7.3 some statistics about which models are available to the user when considering all interactions. We can see that as expected, the *Entity Sentences* and *Neural Question Generator* models are more often present, since they are presented to the user at all times in the conversation, including at the very first turn (while the other models are only fired after the first interaction). As mentioned above, we can also see that the *Topic Classifier* model

User message	***ALICE* possible responses**
Two	Bertrand Russell once said that “it must have required many ages to discover that a brace of pheasants and a couple of days were both instances of the number two”.
I think it would be nice, but I don’t know a whole lot about it	Why do you want nice?. “it” being nice?
We don’t follow prodigies anymore	By “we” do you mean you and me?
I have no idea	“?” No idea about that?
Yes in a way. But I think it’s more an historical article.	Great.. In what way? What does “it” refer to? Next question?. In what way? It depends on what the meaning of the word “it” is. Aha.. In what way? “It” being what?

Table 7.1: Some user messages that fails the *ALICE* model: since the possible responses on the right column contains quotation marks, *ALICE* ignored those valid responses and returned an empty response.

and the *ALICE* model are slightly less presented than the other candidate models. In addition, we clearly notice that the *Simple Answers* model is almost never presented to the user since our set of regular expressions used to fire this model was short and not flexible.

Eventually, it is important to understand, which model is the most often chosen by the human users talking to our chatbot. To that end, we present in diagram (c) of Figure 7.3 the proportion of the number of times each model is selected *when it actually appears in the list of suggested candidate responses.* We see that the model with the highest selection rate is the *Simple Answers* model with a score of 37%. This is reassuring in a sense because even though this model responses are rarely presented to the user, when they are presented, the response we manually defined are preferred by the users. The next most chosen model is the *Entity Sentences* model with a selection rate of 30.81% when available. Already at this stage we can clearly see that the preferred systems according to human users chatting with the bot are rule-based systems. The next two models with comparable selection rates are *DrQA*

and *ALICE* with 21.87% and 21.61% selection rates respectively. The remaining models in decreasing order are *NQG* with 13.69% selection rate, *Fact Retriever* with 10.19% selection rate, and *HRED* reddit, *HRED* twitter, and *Topic Classifier* with 5.21%, 4.77% and 4.71% selection rates respectively. This analysis shows that current fully generative seq-to-seq models are not as good as more restrictive, rule-based, or even retrieval-based models. Although the least often selected model is the *Topic Classifier*, we expect this retrieval-based system to be less popular simply because it is less often relevant to the conversation. Indeed we can expect that if the user asks about the topic of the article and selects the *Topic Classifier* candidate response, they probably will not select it another time in the same conversation.

7.2 Experimentation and Evaluation

After describing and analyzing in detail the dataset we collected from *Amazon Mechanical Turk*, we now present the different experiments we ran on this data and report results on a held-out test set. We want to have a mechanism that can automatically select which response to return to the user given a set of previously generated candidate responses. In Chapter 6 we presented two different neural network architectures (one using hand-crafted features – Figure 6.1, the other using Gated Recurrent Unit networks to automatically extract features – Figure 6.3), two different training algorithms (one using the cross-entropy classification loss – Algorithm 1, the other using the Huber loss with fitted Q-iteration – Algorithm 2), and a rule-based selection process (Figure 6.4). We combine all these different ideas in the following set of experiments:

- **SmallR**: We train the small feed-forward network with hand-crafted features to predict the immediate reward of a given candidate response: either 0 or 1. We thus use Algorithm 1.

- **DeepR**: We train the larger Gated Recurrent Unit network (GRU) to predict the immediate reward of a given candidate response: either 0 or 1. We thus use Algorithm 1 but with the GRU architecture (Figure 6.3).

- **SmallQ**: We train the small feed-forward network with hand-crafted features to predict the Q-value of a given candidate response. We thus use Algorithm 2 but with the architecture described in Figure 6.1.

- **DeepQ**: We train the larger Gated Recurrent Unit network to predict the Q-value of a given candidate response. We thus use Algorithm 2.

All these experiments yield a scoring model that is able to score candidate responses. We then explore three different selection mechanisms:

- **Rule-Based**: We select the response to return to the user by following hand-crafted rules as described in Section 6.2 and in Figure 6.4.

- **Sampled**: We sample (without replacement[2]) a random candidate response according to the distribution given by the different scores given to all candidate responses according to a trained scoring model.

- **Argmax**: We select (without replacement) the candidate response with the highest score according to a trained scoring model.

We split the dataset described above in 80% training set, 10% validation set, and 10% testing set according to the *SQuAD* article's paragraphs. Thus we ensure that both the validation and the testing set have no overlapping article or conversation with the training set. This gives us $1,330$ unique articles in the training set (making $56,564$ examples), 165 in the validation set (making $7,114$ examples), and 168 in the testing set (making $7,083$ examples). From the $70,761$ examples reported in table (a) of Figure 7.2, only $10,292$ have positive reward (+1), while $60,469$ have negative

[2]We sample k responses when evaluating the model with *Recall@k*.

	Entire data	Training set (regular)	Training set (over-sampled)	Validation set	Testing set
unique articles	1,663	1,330	1,330	165	168
all examples	70,761	56,564	96,659	7,114	7,083
positive examples	10,292	8,233	48,328	1,031	1,028
negative examples	60,469	48,331	48,331	6,083	6,055

Table 7.2: Statistics on the regular training set, over-sampled training set, validation set and testing set.

reward (0). This is due to the fact that for each interaction between the chatbot and a user, we collected both the selected candidate responses (labeled as positive examples) and the non-selected candidate responses (labeled as negative examples). Since the user could only select one response and the average number of candidate responses per interaction is 7 (see table (a) in Figure 7.2), we have around 6 times more negative examples than positive ones. Therefore, we make a second version of the training set with over-sampled positive examples. Details of the different training, validation, and testing sets can be seen in Table 7.2. We use this over-sampled training set in all experiments regarding the immediate classification of candidate responses (*SmallR* and *DeepR* experiments). We experiment with both over-sampled and regular training sets for the estimation of Q-values (*SmallQ* and *DeepQ* experiments).

For all our experiments, we explore different combinations of the following parameters by randomly sampling values for each of those 100 times:

- optimizer: *ADAM* (Kingma and Ba, 2014), *SGD* (Rumelhart, G. E. Hinton, and R. J. Williams, 1985; Rumelhart, G. E. Hinton, and R. J. Williams, 1986), *Adagrad* (Duchi, Hazan, and Singer, 2011), *Adadelta* (Zeiler, 2012), and *RMSProp* (G. Hinton, Srivastava, and Swersky, 2012).

- learning rate: 0.01, 0.001, and 0.0001.

- activation function: *Sigmoid* (Equation 3.3), *ReLU* (Glorot, Bordes, and Bengio, 2011), and *pReLU* (K. He et al., 2015).

- dropout rate: 0.2, 0.4, 0.6, and 0.8

We vary only these parameters to limit the number of degrees of freedom in our system, and because we expect that these parameters have a direct influence on the training behavior of our system. We keep the architecture of our networks fixed because we expect the size of the networks to only influence the capacity of our models rather than their ability to learn.

For the reward classification scorers (*SmallR* and *DeepR* experiments), we search the best parameter combination by evaluating the F1 score on the validation set with early stopping and a patience of 20 epochs. We stop training based on the validation F1 score rather than the validation accuracy because of the imbalance we have in the data. Indeed, always classifying an example as negative without learning anything would give us a strong validation accuracy of $\frac{6{,}083}{7{,}114} = 85.51\%$ as we can see in Table 7.2. On the other side, in such cases, the F1 score would be zero as it also takes into account the true positive examples. More specifically we have:

$$F1 = \frac{2 * TP}{2 * TP + FP + FN}, \tag{7.1}$$

with TP being the number of examples we correctly classified as positive, FP being the number of examples we classified as positive but that were negative, and FN being the number of examples we classified as negative but that were positive.

After running 100 *SmallR* experiments and another 100 *DeepR* experiments with different random parameter combinations as described above, the *SmallR* and *DeepR* models with highest F1 validation score were trained with a batch size of 128, the *RmsProp* (G. Hinton, Srivastava, and Swersky, 2012) and *SGD* (Section 3.3.2) optimizers respectively, a learning rate of 0.0001 and 0.001 respectively, $pReLU$ activation functions with *He* weight initialization (K. He et al., 2015) and a dropout rate of 0.2 and 0.4 respectively. This combination of parameters gave us a validation F1 score of 0.420 for the best *SmallR* model, and a validation F1 score of 0.373 for the best *DeepR* model.

For the Q-value estimation scorers (*SmallQ* and *DeepQ* experiments), we searched the best parameter combination by evaluating the Huber loss on the validation set with early stopping and a patience of 20 epochs. Note that in this case we do not classify candidate responses, we only estimate their Q-values. As such, we do not have metrics like accuracy or F1 scores to determine early stopping, this is why we consider the minimum validation loss instead.

After running 100 *SmallQ* experiments and another 100 *DeepQ* experiments with different random parameter combinations as described above, the *SmallQ* and *DeepQ* models with lowest validation loss were trained with the regular training set (i.e.: not the over-sampled one) a batch size of 128, the *SGD* (Section 3.3.2) and *ADAM* (Kingma and Ba, 2014) optimizers respectively, a learning rate of 0.0001, a discount factor of 0.99, an update frequency of $2,000$ updates for the target DQN, a hidden size of 300 for the recurrent networks in *DeepQ* experiments, *sigmoid* activation functions with *Glorot* weight initialization (Glorot and Bengio, 2010) and a dropout rate of 0.8. This combination of parameters gave us a minimal validation loss of 0.00488 for the best *SmallQ* model, and a minimal validation loss of 0.00505 for the best *DeepQ* model.

To automatically evaluate how each of the above models perform, we look at how well they can imitate human selection behaviors on the held-out testing set. Similarly to the *Next Utterance Classification* task (Lowe et al., 2016), we measure the success rate of picking the correct next response; more specifically, we measure *Recall@k* (*R@k*), which is the percentage of correct responses (i.e. the actual response that was chosen by the human) that are found in the top k responses with the highest ranking according to the scorer model. As mentioned above, we consider three different selection mechanism: *Rule-Based*, *Argmax*, and *Sampled*.

We start by evaluating our baseline system on the held-out test set, that is, the scoring network described in Section 6.1.1 and trained for the *ConvAI* challenge with a validation accuracy of 64% on the *ConvAI* competition dataset. This model allowed

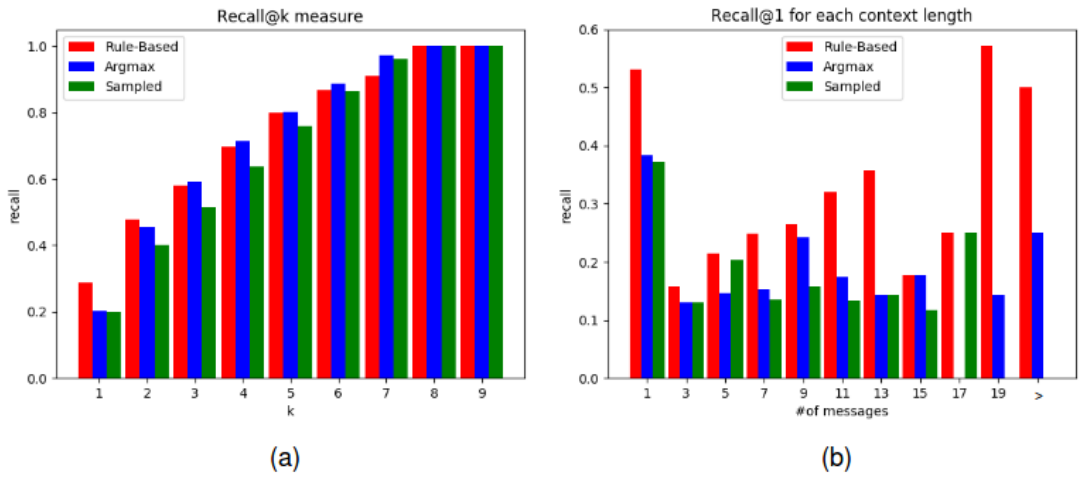


Figure 7.4: Recall measurements for different selection mechanism with our baseline model used during the *ConvAI* challenge. (a) *Recall@k* for all possible values of k. (b) *Recall@1* for different context lengths. The number of candidate responses vary between 2 and 9 with a high peak at 8 as described previously in diagram (c) of Figure 7.2.

us to qualify for the final round by ranking 3rd, however due to narrow hardware restrictions on the final round, our final submission did not perform as well because it was using too much memory (50Gb of RAM). Measurements of the *Recall@k* metric can be seen in diagram (a) of Figure 7.4. We can see that hand-crafted rules described in Section 6.2 helped a lot to boost the *Recall@1* score for this model, jumping *R@1* from 20% with *Argmax* selection to 29% with *Rule-Based* selection. This is a good sign for our hand-crafted rules, as it shows that the assumptions made when designing the rules tend to be correct. Although we only care about *Recall@1* in practice, in is interesting to see that as the number of '*tries*' the chatbot has to recover the correct response increases, the *Rule-Based* selection mechanism does not help much the scoring mechanism, to the point where the *Argmax* selection become better at and after *R@3*. Furthermore, in diagram (b) from Figure 7.4 we represent the *R@1* score for different context lengths. As expected, the *Rule-Based* selection mechanism yields better scores than the other two. We note that for contexts of length 1, the number of candidate responses is only two, thus a random selection would give us a score of

50%. Knowing this, we can see that only the *Rule-Based* selection mechanism (which actually behaves randomly for the first interaction – see Section 6.2) is worth having with a slightly better than random *R@1* score of 53%. In all other cases of context lengths, the number of candidate responses is mostly 8. We can see that in those cases, as the conversation goes on (the number of messages per context increases), the recall score seems to increase as well, up until 13 messages in the context. This is specifically true for the *Rule-Based* selection mechanism, reaching a *R@1* score of 36% in conversations with 13 messages in the context. This can be due to the fact that with longer conversations, our system has more information about the nature of the conversation and is thus better suited to select which candidate response is the most appropriate. Longer contexts (more than 13 messages) are not frequent enough in our dataset to have a meaningful discussion about the *R@1* score.

We present in Figure 7.5 different recall measurements for the best *SmallR* and *DeepR* models with a maximal validation F1 score of 0.420 and 0.373 respectively. These two models were trained to classify candidate responses by minimizing the cross-entropy loss between their predicted values and the true value (either *selected*, or *not*) of the input candidate message. The first thing we notice is the improvement with the baseline model presented earlier from a 29% *R@1* score to a 40% *R@1* score for the best *SmallR* model, and to a 38% *R@1* score for the best *DeepR* model. In addition we can see from diagrams (a) and (c) that the *Argmax* selection mechanism is as good if not better than our custom *Rule-Based* selection mechanism for all values of k in *R@k* scores. This means these models are not only better than the previous ones, they are also much more flexible as they do not require any human crafted rules to select which message to return to the user. Simply returning the message with the highest predicted score yields an *R@1* score of 40% for the best *SmallR* model and of 38% for the best *DeepR* model. Diagrams (a) and (c) also show us that the best *SmallR* model and the best *DeepR* model are similar in terms of overall *Recall@k* score as their score grows at the same rate as k increases. However, the *SmallR* model is

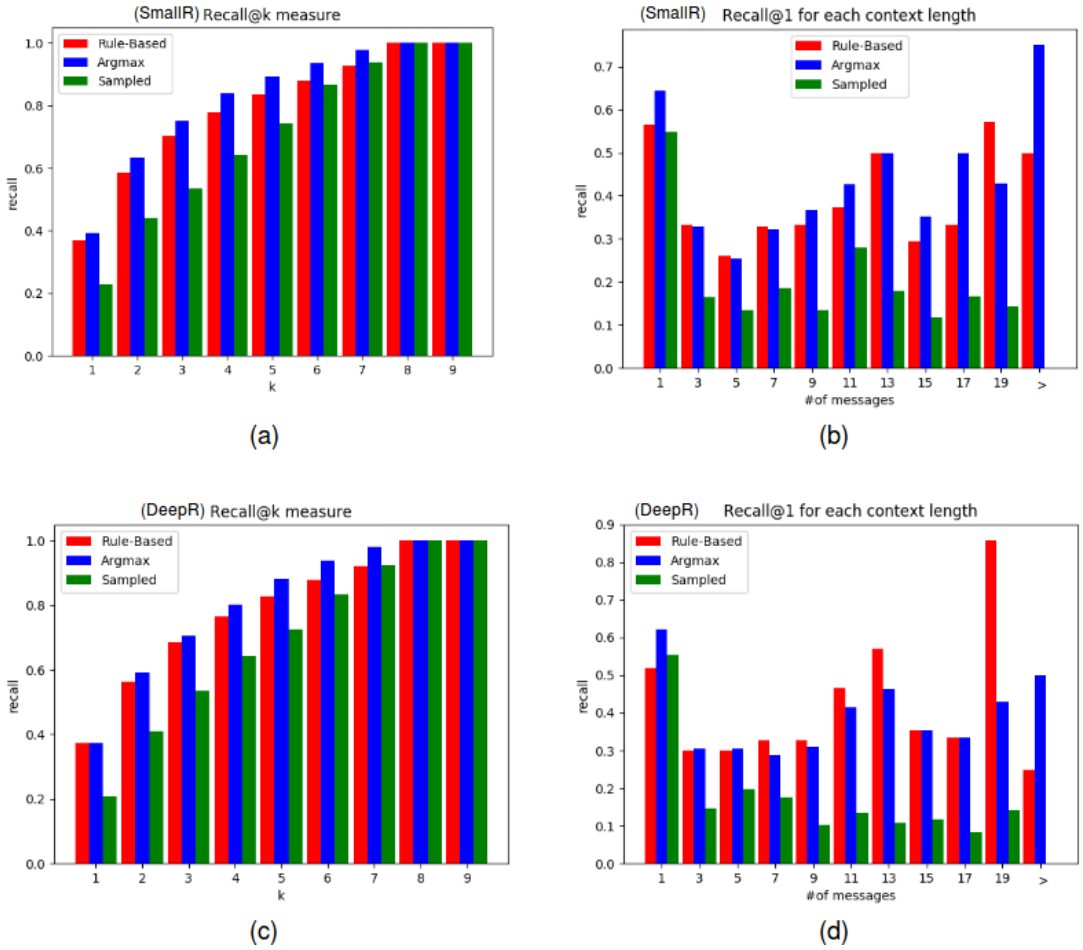


Figure 7.5: Recall measurements for different selection mechanisms with the best *SmallR* and *DeepR* models. (a) *Recall@k* for all possible values of k with the best *SmallR* model. (b) *Recall@1* for different context lengths with the *SmallR* model. (c) *Recall@k* for all possible values of k with the best *DeepR* model. (d) *Recall@1* for different context lengths with the *DeepR* model. The number of candidate responses vary between 2 and 9 with a high peak at 8 as described previously in diagram (c) of Figure 7.2.

slightly better, specially with the *Argmax* selection mechanism which has an average Recall score (computed by taking the average *R@k* score over all values of k) of 77.42% against 75.29% for the best *DeepR* model with *Argmax* selection. This shows us that the deeper architecture involving GRU networks tried to capture meaningful information about the state of the conversation, but our hand-crafted features are still slightly better in those experiments. Furthermore, the deeper architecture being designed more specifically for predicting Q-values by decomposing state and action values, we can expect that this complication may not be optimal for the classification task. Eventually, we report in diagrams (b) and (d) the *R@1* score of both models with different selection mechanisms at different time steps in the conversation. These help us understand how each selection process behaves. Indeed, we can see that the main advantage of the *Argmax* selection over the *Rule-Based* selection we reported from diagram (a) and (c) happens at the beginning of the conversation when the context length is 1. In those cases, the number of candidate responses to choose from is only 2 and the *Rule-Based* selection behaves exactly randomly for the first interaction (as discussed in Section 6.2), it is thus expected that a more informed selection process like *Argmax* or even *Sampled* yields better recall. However, when the discussion contains more messages (i.e.: the context length is greater than 1), the *Rule-Based* selection mechanism is often slightly better than the *Argmax* selection. In general though, as it was the case with our baseline model, we can see that as the context length increases, the *R@1* score also increases up until 13 messages. Longer conversations are not frequent enough in our dataset to have a meaningful discussion about the *R@1* score.

Finally, we present in Figure 7.6 different recall measurements for the best *SmallQ* and *DeepQ* models with a minimal validation loss of 0.00488 and 0.00505 respectively. These two models were trained to predict the expected sum of future return after returning the input candidate response (i.e.: its Q-value) by minimizing the Huber loss between their prediction and the expected sum of future return after considering

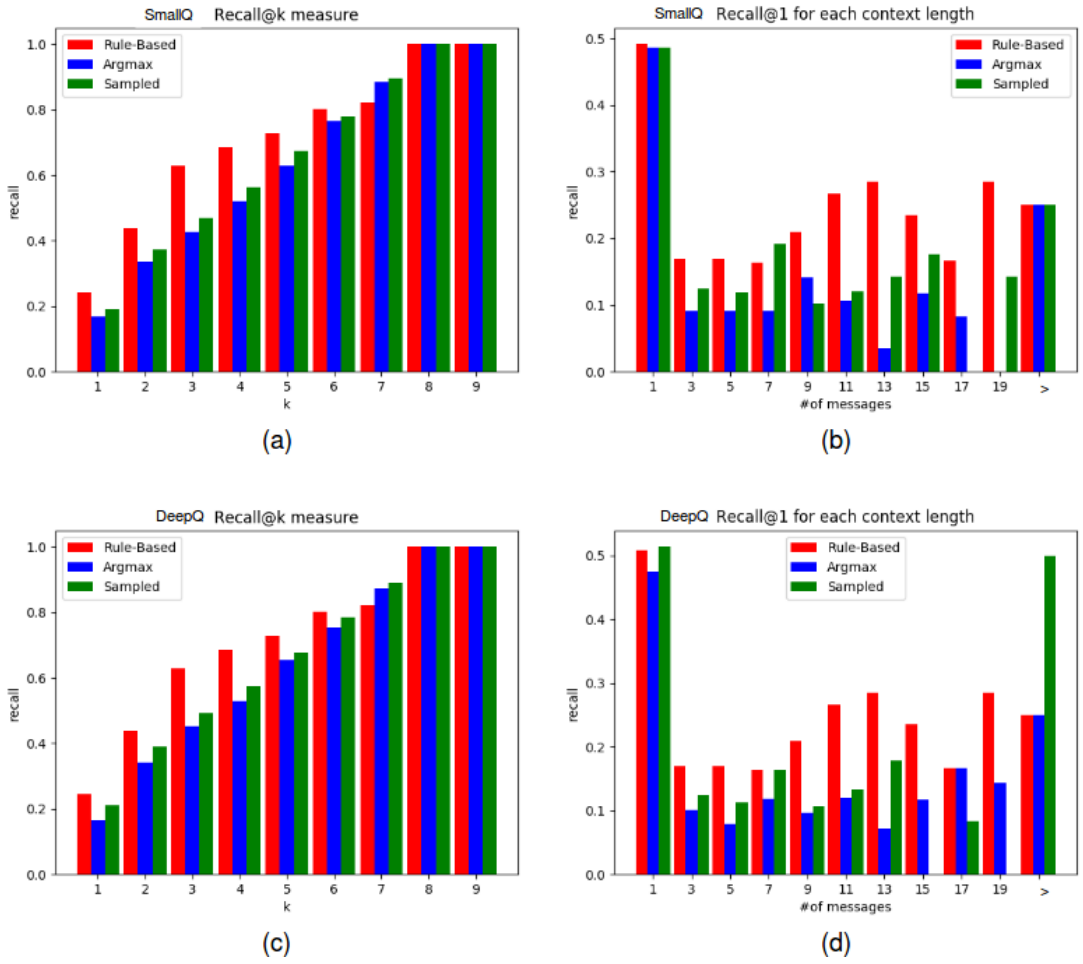


Figure 7.6: Recall measurements for different selection mechanism with the best *SmallQ* and *DeepQ* models. (a) *Recall@k* for all possible values of k with the best *SmallQ* model. (b) *Recall@1* for different context lengths with the *SmallQ* model. (c) *Recall@k* for all possible values of k with the best *DeepQ* model. (d) *Recall@1* for different context lengths with the *DeepQ* model. The number of candidate responses vary between 2 and 9 with a high peak at 8 as described previously in diagram (c) of Figure 7.2.

the input candidate response actual reward. The unfortunate first thing we notice is that these models are actually worst that our simple classifier baseline with the best *R@1* score of 25% attained by the best *DeepQ* model, against our 29% baseline score. The scoring mechanism being poor at evaluating candidate responses we can see from diagrams (a) and (c) that the *Rule-Based* selection process is stronger than the other two for all values of k under 7, after which each selection mechanism is performing similarly given that the number of candidate responses is 8 most of the time. Overall, the average Recall score of the *SmallQ* model is slightly lower than its counter part *DeepQ*, specially with the *Rule-Based* selection mechanism which has an average Recall score (computed by taking the average *R@k* score over all values of k) of 62.08% against 63.12% for the best *DeepQ* model with *Rule-Based* selection. This shows us that the deeper architecture involving GRU networks is preferred to estimate Q-values. This is to be expected as the architecture was inspired by *dueling Deep Q-Networks* (Wang et al., 2015). Another interesting result we can observe is that, unlike in the classifier models (Baseline, *SmallR*, and *DeepR*), the *Sampled* selection process seems to perform better than the *Argmax* selection in both *SmallQ* and *DeepQ* models. This shows that being greedy with respect to the predicted Q-value may not always be the best strategy, and allowing some stochasticity can be beneficial in those cases. This is another sign that the predicted Q-values are not informative enough to make a greedy decision. Eventually, if we look at diagrams (b) and (d) we can still see that, just like in our previous experiments, longer contexts allow better *R@1* score with the *Rule-Based* selection mechanism.

Overall, the experiments conducted and described above show us that the additional data is indeed informative about how humans pick their responses, and can be used to train models that can make better decision while minimizing the need of human expertise by automatically extracting text features. On the other hand, we clearly see that the choice of the optimization algorithm used is critical as our *SmallQ* and *DeepQ* experiments yielded poor results. This can be caused by the fact that

the state and action space possible in our conversational environment is enormous if not infinite, and that our collected data cannot possibly capture all of it. It is often the case that Deep Q-Networks are trained in simulated environments, allowing the model to explore a lot of different trajectories. Unfortunately this remains a challenge in the dialog task. A conversation often being symmetric, having such a simulator is as hard as solving the original task. On the other hand, training an agent while talking with real users can become impractical given the time response of real users, their personal patience to exploratory behaviors from our chatbot, and their adaptive evaluation as the chatbot would become better over time.

After running all the previously described experiments, we took the best combination of ranking and selection mechanism (ie: the *SmallR* ranking and the *Argmax* selection with an average $R@k$ score of 77.42%) and collected a few conversations with human users. We present in Table 7.3 four conversations: two good and coherent, and two incoherent. The top left conversation follows a question – answer structure with the chatbot asking most of the questions with the *Entity Sentences* model. At the end of the conversation the bot is then answering user questions using the *DrQA* model. The top right conversation is more of a social chat and the topic of the conversation diverges a little from the article but stays coherent. These first two conversations are some good examples of coherent interactions we collected. The lower left chat is an example in which the chatbot changes topic with the *Fact Retriever* model. In addition, we can see that the agent contradicts itself in the last interaction with the "I'm a king." reply. The lower right conversation is an example in which our chatbot goes in circle and no progress is made in the chat. These two conversations are examples of incoherent interactions we collected with our dialog agent. One weakness of using probabilistic selection mechanism (*Argmax*) is that we do not explicitly check for contradicting information or for repetitions from our chatbot, whereas a simple rule based system could avoid those issues. On the other hand, we argue that using probabilistic models allow the system to be more flexible to new conversation topics.

article	<...>. Dou Wu and the Grand Tutor Chen Fan attempted a coup d'état against the eunuchs Hou Lan, Cao Jie, and Wang Fu. <...> General Zhang Huan favored the eunuchs. <...>	<...>. Most of these companies <...> temporarily disable the 10NES chip in the NES. <...>
bot	Do you know Wu in person?	Do you think you would like to work for NES?
user	No, I do not.	Yes I would.
bot	Did you know that A Boeing 747s wingspan is longer than the Wright brother's first flight.?	The article is related to Computer & Internet.
user	I didn't know that.	That is interesting.
bot	Do you know what Chen Fan did in his life?	I'm a fan of the game.
user	He was arrested for an attempted coup d'état.	What game?
bot	I don't have a lot of other questions about this article. Maybe you can ask me one?	We are playing Turing's imitation game.
user	Who favoured the eunuchs?	How does it go?
bot	General Zhang Huan	How should I know how it go does.
article	<... > Margaret Thatcher travelled to Beijing to negotiate with the Chinese government <...>	<...>. The Los Angeles Avengers announced that they were folding the franchise. <...>
bot	Are you a Chinese ?	Have you ever used any of the Los Angeles Avengers's products or services?
user	I am not chinese I'm american	No i have not
bot	i like the way you think . i like it .	Are you kidding? Do you think you will?
user	That's just my brain	Is it any good?
bot	Do you know that In ancient Greece idiot meant a private citizen or layman.?	This article is about Sports
user	Ya those silly philosophers	Which one?
bot	I'm a Romanian and I'm not even a king .	The article is related to Sports
user	Well there aren't very many kings nowadays	Basketball?
bot	I'm a king .	I think it's about Sports

Table 7.3: Examples of four bot-to-human conversations with our final chatbot, using the best *SmallR* ranker and the *Argmax* selection mechanism.

8

Conclusion

–C'est le langage qui crée l'homme.

–It is the language that creates the man.

Jean D'Ormesson

After presenting different experiments for our ranking and selection mechanisms, we conclude our work in this final chapter by providing a short summary, highlighting some limitations and mentioning some future work.

8.1 Summary

Throughout this thesis we presented the *RLLChatBot*: a conversational agent capable of discussing about random news paragraphs with a human user. Using an ensemble of classical rule-based models and flexible modern techniques, the chatbot we created differentiates itself from previous conversational agents in many ways. Being non-goal oriented, it has to be flexible enough to discus a wide range of topics, which motivated our use of different models ranging in their specificity. We use nine models to generate up to nine distinct candidate responses at each interaction of a conversation. Three of them are fully generative, relying on the encoder-decoder sequence-to-sequence architecture. Three of them are retrieval-based models that either select a text span from the news paragraph, give a topic to the paragraph, or return a relevant fact. The remaining three models are rule-based systems that try to capture text entities

present in the news paragraph or in the conversation to produce meaningful responses. In addition, the production of a response from our chatbot is done in two steps: we first generate multiple candidate responses with the nine available models, before selecting the best one according to a trained ranking mechanism. In contrast, typical conversational agents use at least 3 modules to produce a response: a natural language understanding machine, a dialog manager (often made of many sub-modules), and a natural language generator.

Another focus of this work is the presentation of a novel dataset collected to train different ranking mechanisms, and the comparison of their strengths and weaknesses. We explore two distinct learning algorithms: one relying on supervised learning to perform a classification task, the other relying on reinforcement learning to perform a prediction task. We also consider two types of architectures: one using hand-crafted features with a feed-forward neural network, the other using a deeper architecture involving automatic feature extraction with Gated Recurrent Unit (GRU) networks connected to a feed-forward network. We find that the difference between the deep GRU network and the simpler feed-forward network with hand-crafted features is negligible, thus permitting to use the deeper architecture in order to have a more flexible system (not relying on human expertise) at the expense of training time. However, we find the training algorithm to be a critical decision as the models trained to predict Q-values with reinforcement learning techniques are not as powerful as the baseline model according to the *Recall@k* metric. On the other hand, models trained to classify candidate responses in a supervised fashion suffer from an imbalance training set, but after oversampling positive examples we make a significant improvement on the *Recall@1* score compared to the initial baseline model we used for the *ConvAI* challenge by going from a *Recall@1* of 29% to 40% with the best selection mechanism shifting from our *Rule-Based* mechanism to the more flexible *Argmax* selection.

8.2 Limitations

We now present a few limitations of this work. Being partly motivated by an organized competition, we had some time constraints that did not allow us to always pursue all the experiments planned.

The first set of limitations comes with the different set of generative models described in Chapter 5. One obvious extension would be with regards to the two *HRED* models (Section 5.1.1), more specifically the one trained on *Reddit* data: since we want the model to chat about a given news paragraph, it would have been beneficial for this model to also learn to encode the article that triggered each *Reddit* discussion in the training set. That is, instead of conditioning the decoder on the conversation history only, we could also condition it on the encoded article, after being processed by a recurrent neural network. In addition, instead of vanilla *HRED* models, we believe that adding an *Attention* mechanism (Bahdanau, Cho, and Bengio, 2014) might help the model generate less generic responses. Another model that we believe could have been improved is the *Topic Classifier* model (Section 5.2.2). As of now, we use a simple 10-class classifier and some predefined sentences to inform the user about the general topic. However, there is a lot of work done in the area of text summarizarion that could be explored. For cases in which the news paragraph is quite long, a summarization model could have been beneficial. There exists two types of summarization: one is extractive (copying text spans from the source text to build a summary – Cheng and Lapata, 2016), the other is abstractive (much more challenging, the model generates novel sentences to build a summary of the source text – Rush, Chopra, and Weston, 2015). These are experiments that we did not have time to pursue before the deadline for the *ConvAI* challenge in December 2017. After the competition, we focused on building better ranking and selection mechanisms, thus keeping the generative models fixed.

Regarding the different scoring techniques used, one limitation is that the deep

architectures presented, involving Gated Recurrent Unit (GRU) networks, were not pre-trained on large corpora of text. In our small dataset regime, we believe that training the recurrent networks to encode and decode conversational text (just like the *HRED* models) would have been beneficial for the scoring models. Indeed, after pre-training the recurrent networks, we expect them to better capture important features that are used to predict the next sentence (as a language modelling model would perform) and that can be beneficial to also score candidate responses. Furthermore, we also acknowledge that the deep architecture presented in Section 6.1.2 is better suited for predicting Q-values as it clearly splits the *state* (context & article) value and the advantage of taking a given *action* (candidate response) in the current conversation. We believe that different architectures that do not explicitly split the data like so may yield better results in the classification task presented in Section 6.1.1.

8.3 Future Work

Eventually, we leave as future work the difficult task of unifying the presented pipeline chatbot into an end-to-end trainable system. As previously mentioned, the nine generative systems producing candidate responses were trained once before the *ConvAI* competition and remained fixed thereafter. After focusing on the different scoring mechanisms, even if we get a *Recall@1* score of 100%, the chatbot would still be limited by the capabilities of its components. One way to remedy that would be to alternate phases of data-collection with the current chatbot by asking humans to chat and evaluate responses; fine-tuning a pre-defined scoring machine by training it on the latest conversational data; and fine-tuning generative models by trying to produce responses that maximize the score given by our most recent version of the scoring machine. This can be achieved using *Policy Gradient* algorithms (Sutton, McAllester, et al., 2000) on end-to-end systems such as *HRED* models. After being pre-trained to generate the next utterance in a dialog, these models can be fine-tuned to produce

responses that maximize the score given by the previously trained scoring machine. After convergence, we can go back and collect more data with the newest generative model, re-train a new version of the scoring machine, and fine-tune again the generative model to maximize the score given by our latest scoring machine. We expect this recurrent procedure to produce better and better responses from our end-to-end generative models, and to be a first step towards having a model that can return the appropriate response in *one* step.

Finally, participating in the *ConvAI* competition allowed us to thoroughly assess and benchmark our initial system. Organizing academic competitions like the *Amazon Alexa Prize* and the *ConvAI* challenge are good alternatives to evaluate conversational agents in various tasks. Our work shows that making the data available can be useful to drive dialog research. Indeed, by collecting more data in the same fashion as in the competition, we were able to improve our scoring machine. We believe that future challenges should thus encourage participation to have a good amount of evaluated conversational data, and release the data after the end of the competition. From our experience, we also encountered several engineering difficulties to deploy our system in a live environment with several participants talking to it. Often taking more time than expected, these challenges can reduce the amount of innovation in a system. We thus believe that future academic challenges should provide as much help as possible to deploy systems in an easy and secure fashion. One way that may help is to make an engineer available for helping teams deploying their solutions. We acknowledge that engineering skills are mandatory for real products, but since the goal of academic competitions is to get a sense of where the research field is and where it is going, these skills should becomes less important in those scenarios.

Bibliography

Aust, Harald et al. (1995). "The Philips automatic train timetable information system". In: *Speech Communication* 17.3-4, pp. 249–262.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473*.

Banerjee, Satanjeev and Alon Lavie (2005). "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments". In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pp. 65–72.

Barker, Jon et al. (2013). "The PASCAL CHiME speech separation and recognition challenge". In: *Computer Speech & Language* 27.3, pp. 621–633.

Bellman, Richard (1957). "A Markovian decision process". In: *Journal of Mathematics and Mechanics*, pp. 679–684.

Bengio, Yoshua, Réjean Ducharme, et al. (2003). "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb, pp. 1137–1155.

Bengio, Yoshua, Paolo Frasconi, and Patrice Simard (1993). "The problem of learning long-term dependencies in recurrent networks". In: *Neural Networks, 1993., IEEE International Conference on*. IEEE, pp. 1183–1188.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Bradeško, Luka and Dunja Mladenić (2012). "A survey of chatbot systems through a loebner prize competition". In: *Proceedings of Slovenian Language Technologies Society Eighth Conference of Language Technologies*, pp. 34–37.

Callejas, Zoraida and Ramón López-Cózar (2005). "Implementing modular dialogue systems: A case of study". In: *COST278 Final Workshop and ITRW on Applied Spoken Language Interaction in Distributed Environments*.

Chen, Danqi et al. (2017). "Reading Wikipedia to Answer Open-Domain Questions". In: *Association for Computational Linguistics (ACL)*.

Cheng, Jianpeng and Mirella Lapata (2016). "Neural summarization by extracting sentences and words". In: *arXiv preprint arXiv:1603.07252*.

Cho, Kyunghyun, Bart Van Merriënboer, Dzmitry Bahdanau, et al. (2014). "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259*.

Cho, Kyunghyun et al. (2014a). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*.

— (2014b). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*.

Chung, Junyoung et al. (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555*.

Colby, Kenneth (1975). *Artificial Paranoia: A Computer Simulation of Paranoid Process.* Pergamon Press, New York.

Cunningham, I., S.L. Pache, and C. White (2007). *Customer service messaging, such as on mobile devices*. EP Patent App. EP20,050,788,755.

Danescu-Niculescu-Mizil, Cristian, Michael Gamon, and Susan Dumais (2011). "Mark my words!: linguistic style accommodation in social media". In: *Proceedings of the 20th international conference on World wide web*. ACM, pp. 745–754.

Du, Xinya, Junru Shao, and Claire Cardie (2017). "Learning to ask: Neural question generation for reading comprehension". In: *arXiv preprint arXiv:1705.00106*.

Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul, pp. 2121–2159.

Dušek, Ondřej and Filip Jurčíček (2016). "Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings". In: *arXiv preprint arXiv:1606.05491*.

Elman, Jeffrey L (1990). "Finding structure in time". In: *Cognitive science* 14.2, pp. 179–211.

Epstein, Robert (1993). *Loebner Prize Competition in Artificial Intelligence: Official Transcripts and Results*. Tech. rep. Cambridge Center for Behavioral Studies.

Ernst, Damien, Pierre Geurts, and Louis Wehenkel (2005). "Tree-based batch mode reinforcement learning". In: *Journal of Machine Learning Research* 6.Apr, pp. 503–556.

Ferrucci, David et al. (2010). "Building Watson: An overview of the DeepQA project". In: *AI magazine* 31.3, pp. 59–79.

Forgues, Gabriel et al. (2014). "Bootstrapping dialog systems with word embeddings". In: *Nips, modern machine learning and natural language processing workshop*. Vol. 2.

Fryer, LK and Rollo Carpenter (2006). "Bots as language learning tools". In: *Language Learning & Technology*.

Gers, Felix A and Jürgen Schmidhuber (2000). "Recurrent nets that time and count". In: *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*. Vol. 3. IEEE, pp. 189–194.

Gers, Felix A, Jürgen Schmidhuber, and Fred Cummins (1999). "Learning to forget: Continual prediction with LSTM". In:

Gilbert, CJ Hutto Eric (2014). "Vader: A parsimonious rule-based model for sentiment analysis of social media text". In: *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf.*

Girshick, Ross (2015). "Fast r-cnn". In: *arXiv preprint arXiv:1504.08083*.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep sparse rectifier neural networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323.

Goodfellow, Ian et al. (2016). *Deep learning*. Vol. 1. MIT press Cambridge.

Gordon, Geoffrey J (1995). "Stable function approximation in dynamic programming". In: *Machine Learning Proceedings 1995*. Elsevier, pp. 261–268.

Gorin, Allen L, Giuseppe Riccardi, and Jeremy H Wright (1997). "How may I help you?" In: *Speech communication* 23.1-2, pp. 113–127.

Hakkani-Tür, Dilek et al. (2016). "Multi-Domain Joint Semantic Frame Parsing Using Bi-Directional RNN-LSTM." In: *Interspeech*, pp. 715–719.

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

Hebb, Donald Olding (1949). *The organization of behavior: A neuropsychological theory*. Psychology Press.

Henderson, Matthew, Blaise Thomson, and Jason D Williams (2014a). "The second dialog state tracking challenge". In: *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pp. 263–272.

Henderson, Matthew, Blaise Thomson, and Jason D Williams (2014b). "The third dialog state tracking challenge". In: *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, pp. 324–329.

Henderson, Matthew, Blaise Thomson, and Steve Young (2013). "Deep neural network approach for the dialog state tracking challenge". In: *Proceedings of the SIGDIAL 2013 Conference*, pp. 467–471.

Hinton, Geoff, N Srivastava, and K Swersky (2012). "Neural networks for machine learning. lecture 6e." In: *University of Toronto, csc321*.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hopfield, John J (1982). "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences* 79.8, pp. 2554–2558.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5, pp. 359–366.

Hu, Zhiting et al. (2017). "Toward controlled generation of text". In: *International Conference on Machine Learning*, pp. 1587–1596.

Hutchens, Jason L and Michael D Alder (1998). "Introducing megahal". In: *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, pp. 271–274.

Jordan, Michael (1986). "Attractor dynamics and parallelism in a connectionist sequential machine". In: *Eighth Annual Conference of the Cognitive Science Society, 1986*, pp. 513–546.

Joulin, Armand et al. (2016). "Bag of tricks for efficient text classification". In: *arXiv preprint arXiv:1607.01759*.

Kay, Timothy and Robert Hoffer (2007). *Method and system for using screen names to customize interactive agents*. US Patent 7,266,585.

— (2006). *Method for performing authenticated access to a service on behalf of a user*. US Patent 7,146,404.

Kim, Seokhwan, Luis Fernando D'Haro, et al. (2016). "The fifth dialog state tracking challenge". In: *Spoken Language Technology Workshop (SLT), 2016 IEEE*. IEEE, pp. 511–517.

Kim, Seokhwan, Luis Fernando D'Haro, et al. (2017). "The fourth dialog state tracking challenge". In: *Dialogues with Social Robots*. Springer, pp. 435–449.

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization*.

Kurata, Gakuto et al. (2016). "Leveraging sentence-level information with encoder lstm for semantic slot filling". In: *arXiv preprint arXiv:1601.01530*.

Lee, Byung-Jun and Kee-Eung Kim (2016). "Dialog History Construction with Long-Short Term Memory for Robust Generative Dialog State Tracking". In: *Dialogue & Discourse* 7.3, pp. 47–64.

Lee, Ji Young and Franck Dernoncourt (2016). "Sequential short-text classification with recurrent and convolutional neural networks". In: *arXiv preprint arXiv:1603.03827*.

Lester, James, Karl Branting, and Bradford Mott (2004). "Conversational agents". In: *The Practical Handbook of Internet Computing*, pp. 220–240.

Levin, Esther and Michael Fleisher (1988). "Accelerated learning in layered neural networks". In: *Complex systems* 2, pp. 625–640.

Li, Jiwei, Michel Galley, Chris Brockett, Jianfeng Gao, et al. (2015). "A diversity-promoting objective function for neural conversation models". In: *arXiv preprint arXiv:1510.03055*.

Li, Jiwei, Michel Galley, Chris Brockett, Georgios P Spithourakis, et al. (2016). "A persona-based neural conversation model". In: *arXiv preprint arXiv:1603.06155*.

Li, Jiwei, Will Monroe, et al. (2016). "Deep reinforcement learning for dialogue generation". In: *arXiv preprint arXiv:1606.01541*.

Li, Xiujun, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz (2017). "Investigation of Language Understanding Impact for Reinforcement Learning Based Dialogue Systems". In: *arXiv preprint arXiv:1703.07055*.

Li, Xuijun, Yun-Nung Chen, Lihong Li, and Jianfeng Gao (2017). "End-to-end task-completion neural dialogue systems". In: *arXiv preprint arXiv:1703.01008*.

Lin, Chin-Yew (2004). "Rouge: A package for automatic evaluation of summaries". In: *Text Summarization Branches Out*.

Lin, Long-Ji (1992). "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine learning* 8.3-4, pp. 293–321.

Lipton, Zachary C., John Berkowitz, and Charles Elkan (2015). "A Critical Review of Recurrent Neural Networks for Sequence Learning". In: *arXiv preprint arXiv:1506.00019*.

Littman, Michael L (1994). "Markov games as a framework for multi-agent reinforcement learning". In: *Machine Learning Proceedings 1994*. Elsevier, pp. 157–163.

Liu, Bing and Ian Lane (2016). "Attention-based recurrent neural network models for joint intent detection and slot filling". In: *arXiv preprint arXiv:1609.01454*.

Liu, Chia-Wei et al. (2016). *How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation*.

Lowe, Ryan et al. (2016). "On the evaluation of dialogue systems with next utterance classification". In: *arXiv preprint arXiv:1605.05414*.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

McGlashan, Scott et al. (1992). "Dialogue management for telephone information systems". In: *Proceedings of the third conference on Applied natural language processing*. Association for Computational Linguistics, pp. 245–246.

Mesnil, Grégoire et al. (2015). "Using recurrent neural networks for slot filling in spoken language understanding". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23.3, pp. 530–539.

Mikolov, Tomas, Kai Chen, et al. (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Mikolov, Tomáš, Anoop Deoras, et al. (2011). "Strategies for training large scale neural network language models". In: *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, pp. 196–201.

Mikolov, Tomáš, Martin Karafiát, et al. (2010). "Recurrent neural network based language model". In: *Eleventh Annual Conference of the International Speech Communication Association*.

Mikolov, Tomas, Ilya Sutskever, et al. (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*, pp. 3111–3119.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, et al. (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, p. 529.

Mrkšić, Nikola et al. (2016). "Neural Belief Tracker: Data-driven dialogue state tracking". In: *arXiv preprint arXiv:1606.03777*.

Oh, Alice H and Alexander I Rudnicky (2002). "Stochastic natural language generation for spoken dialog systems". In: *Computer Speech & Language* 16.3-4, pp. 387–407.

Papineni, Kishore et al. (2002). "BLEU: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pp. 311–318.

Pennington, Jeffrey, Richard Socher, and Christopher Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Rajpurkar, Pranav et al. (2016). "Squad: 100,000+ questions for machine comprehension of text". In: *arXiv preprint arXiv:1606.05250*.

Raux, Antoine et al. (2005). "Let's Go Public! Taking a spoken dialog system to the real world". In: *Ninth European Conference on Speech Communication and Technology*.

Ravuri, Suman and Andreas Stolcke (2015). "Recurrent neural network and lstm models for lexical utterance classification". In: *Sixteenth Annual Conference of the International Speech Communication Association*.

Riedmiller, Martin (2005). "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *European Conference on Machine Learning*. Springer, pp. 317–328.

Ritter, Alan, Colin Cherry, and Bill Dolan (2010). "Unsupervised modeling of twitter conversations". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 172–180.

Ritter, Alan, Colin Cherry, and William B Dolan (2011). "Data-driven response generation in social media". In: *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, pp. 583–593.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Roy, Nicholas, Joelle Pineau, and Sebastian Thrun (2000). "Spoken dialogue management using probabilistic reasoning". In: *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pp. 93–100.

Rudnicky, Alexander I et al. (1999). "Creating natural dialogs in the Carnegie Mellon Communicator system". In: *Sixth European Conference on Speech Communication and Technology*.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

— (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, p. 533.

Rus, Vasile and Mihai Lintean (2012). "A comparison of greedy and optimal assessment of natural language student input using word-to-word similarity metrics". In: *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, pp. 157–162.

Rush, Alexander M, Sumit Chopra, and Jason Weston (2015). "A neural attention model for abstractive sentence summarization". In: *arXiv preprint arXiv:1509.00685*.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252.

Schuster, Mike and Kuldip K Paliwal (1997). "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681.

Serban, Iulian V., Chinnadhurai Sankar, et al. (2017). *A Deep Reinforcement Learning Chatbot*.

Serban, Iulian Vlad, Alessandro Sordoni, et al. (2016). "Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models." In: *AAAI*. Vol. 16, pp. 3776–3784.

Simpson, Andrew and Norman M Eraser (1993). "Black box and glass box evaluation of the SUNDIAL system". In: *Third European Conference on Speech Communication and Technology*.

Sordoni, Alessandro et al. (2015). "A neural network approach to context-sensitive generation of conversational responses". In: *arXiv preprint arXiv:1506.06714*.

Srivastava, Nitish et al. (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Su, Pei-Hao, Pawel Budzianowski, et al. (2017). "Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management". In: *arXiv preprint arXiv:1707.00130*.

Su, Pei-Hao, Milica Gasic, et al. (2016). "On-line active reward learning for policy optimisation in spoken dialogue systems". In: *arXiv preprint arXiv:1605.07669*.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*, pp. 3104–3112.

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge.

Sutton, Richard S, David A McAllester, et al. (2000). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*, pp. 1057–1063.

Tang, Duyu, Bing Qin, and Ting Liu (2015). "Document modeling with gated recurrent neural network for sentiment classification". In: *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1422–1432.

Turing, Alan M (1950). "Computing Machinery and Intelligence". In: *Mind* 49.49, pp. 433–460.

Van Hasselt, Hado, Arthur Guez, and David Silver (2016a). "Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. Vol. 16, pp. 2094–2100.

— (2016b). "Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. Vol. 16, pp. 2094–2100.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems*, pp. 6000–6010.

Vinyals, Oriol and Quoc Le (2015). "A neural conversational model". In: *arXiv preprint arXiv:1506.05869*.

Wallace, Richard S (2009). "The anatomy of ALICE". In: *Parsing the Turing Test*. Springer, pp. 181–210.

Wang, Ziyu et al. (2015). "Dueling network architectures for deep reinforcement learning". In: *arXiv preprint arXiv:1511.06581*.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8.3-4, pp. 279–292.

Weinberger, Kilian et al. (2009). "Feature hashing for large scale multitask learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, pp. 1113–1120.

Weizenbaum, Joseph (1966). "ELIZA - a computer program for the study of natural language communication between man and machine". In: *Communications of the ACM* 9.1, pp. 36–45.

Wen, Tsung-Hsien, Milica Gasic, Dongho Kim, et al. (2015). "Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking". In: *arXiv preprint arXiv:1508.01755*.

Wen, Tsung-Hsien, Milica Gasic, Nikola Mrksic, et al. (2015). "Semantically conditioned lstm-based natural language generation for spoken dialogue systems". In: *arXiv preprint arXiv:1508.01745*.

Widrow, Bernard and Marcian E Hoff (1960). *Adaptive switching circuits*. Tech. rep. STANFORD UNIV CA STANFORD ELECTRONICS LABS.

Williams, Jason D and Steve Young (2007). "Partially observable Markov decision processes for spoken dialog systems". In: *Computer Speech & Language* 21.2, pp. 393–422.

Williams, Jason, Antoine Raux, and Matthew Henderson (2016). "The dialog state tracking challenge series: A review". In: *Dialogue & Discourse* 7.3, pp. 4–33.

Williams, Jason, Antoine Raux, Deepak Ramachandran, et al. (2013). "The dialog state tracking challenge". In: *Proceedings of the SIGDIAL 2013 Conference*, pp. 404–413.

Young, Steve J (2000). "Probabilistic methods in spoken–dialogue systems". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 358.1769, pp. 1389–1402.

Young, Steve et al. (2013a). "Pomdp-based statistical spoken dialog systems: A review". In: *Proceedings of the IEEE* 101.5, pp. 1160–1179.

— (2013b). "Pomdp-based statistical spoken dialog systems: A review". In: *Proceedings of the IEEE* 101.5, pp. 1160–1179.

Zaremba, Wojciech and Ilya Sutskever (2014). "Learning to execute". In: *arXiv preprint arXiv:1410.4615*.

Zeiler, Matthew D (2012). "ADADELTA: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701*.

Zhang, Xiang and Yann LeCun (2015). "Text understanding from scratch". In: *arXiv preprint arXiv:1502.01710*.

Zue, Victor W and James R Glass (2000). "Conversational interfaces: Advances and challenges". In: *Proceedings of the IEEE* 88.8, pp. 1166–1180.