ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA E SCIENZE INFORMATICHE

End-to-End Goal-Oriented Conversational Agent for Risk Awareness

Thesis in Autonomous Systems

Supervisor:

Prof. Andrea Omicini

Presented by:

Marco Canducci

Company supervisor:

Dott. Mauro Gatti

Session III

Academic Year 2018/2019

Ai miei genitori, le persone più coraggiose che conosca

Table of contents

List of Figures

Chapter 1

Introduction

Traditional development of conversational agents for goal-oriented applications typically require a lot of domain-specific handcrafting, which precludes effectively scaling up to different domains; end-to-end systems would escape this limitation because they can be trained directly from dialogues [1]. The encouraging success recently obtained in end-to-end chit-chat bots could carry over to goal-oriented settings: applying deep learning models for building robust and scalable digital assistants directly from corpora of conversations is in fact still a challenging task and an open research area. For this reason, I decided with my company supervisor that it would have been more relevant in the context of a master's thesis to experiment and get acquainted with new promising methodologies - although not yet ready for production - rather than investing time in hand-crafting dialogue rules for a specific domain.

My internship at the *IBM Research Center on Active Intelligence* located in Bologna spanned from September 2019 to February 2020, period during which I worked on two different projects, both focused on new methodologies for Human-Computer Interaction. One of them regarded my thesis work, and had the following macro objectives:

- investigate the latest scientific studies concerning goal-oriented conversational agents, studying new methods to develop them and, in general, engage users;
- choose a reference study, understand it and implement it with an appropriate technology;
- apply what learnt to a particular domain of interest.

We chose the application domain of *Risk awareness*, a decision taken after the participation to the hackathon organized by DigiEduHack [2] in collaboration with IBM, UniBo and Unipol during the 3rd and the 4th of October.

Figure 1.1: Awarding of our team - *I Mancini* - winner of the hackathon organized by DigiEduHack in collaboration with IBM, UniBo and Unipol. Out of five members, four of us were from the School of Computer Science and Engineering located in Cesena [3].

Specifically, for my thesis project I developed the prototype of a conversational agent aimed to educate and advise users on the topic of risk. We decided that, as a key feature, the system should have been able to learn directly from past conversations, therefore being trainable *end-to-end*. Not having real dialogues available though, I took care of synthetically generate a corpora of conversations between a user and a conversational agent, taking a cue from the *Dialog bAbI dataset* for restaurant reservations and adapting it to the new domain of interest.

Chapter 2

State of the art

2.1 Background

Conversational Agents are virtual entities that communicate with users in natural language (text, speech, or both), and fall into two main categories [4].

Goal-oriented dialogue agents, or *Task*-oriented dialogue agents, use the conversation with the users to help them complete tasks or, in general, to achieve goals; these kind of virtual assistants are typically set up to have short conversations and, depending on the target userbase, they can be implemented to cover broad, generic topics or, alternatively, to focus on narrower domains in order to be more effective in specific areas. Examples of the former are digital assistants like Siri, Alexa, Google Now, which give directions, control appliances, find restaurants, or make calls; the latter provide more ad hoc solutions like answering frequent questions on corporate websites, giving personalized financial advice [5], or even doing social good: *DoNotPay* is the first "robot lawyer" that helps people challenge incorrect parking fines, apply for emergency housing, or claim asylum for refugees [6].

By contrast, **chatbots** or *chit-chat bots*, are systems designed for extended interactions, set up to mimic the unstructured conversations or "chats" characteristic of human-to-human interaction; these systems have historically been studied to pass the Turing test, for pure entertainment (see Microsoft's XiaoIce emotional companion [7]), but also for practical purposes like testing theories of psychological counseling or, lately, making goal-oriented agents more natural.

Goal-oriented dialog agents have been historically built with four major components: a pre-processing unit, a natural language understanding unit, a dialog manager and a response generator [8]. This structure allows to model the conversation flow in detail, providing a strong control to the designer but, at the same time, limiting the agent's capabilities (i) to the ones explicitly foreseen by the developer and (ii) to the specific domain of interest it's being developed for. Since extraordinary results have been obtained with deep learning frameworks for chit-chat bots, the NLP and AI communities are verifying the suitability of these end-to-end solutions for goal-oriented systems, in order to develop dialog agents from data with as little as possible human intervention; furthermore, one long-term objective is to increasingly blur the dividing line between the two dialog systems categories, with the aim to develop virtual agents capable of supporting "chit chat" conversations and, at the same time, effectively helping users achieving specific goals.

2.2 Search objective and strategy

The main objective of this chapter is to identify both the founding studies and the most recent (and promising) ones which describe how to develop goal-oriented dialogue systems trained directly from data. The motivation behind such an investigation is twofold: first of all there's a rapidly growing market demand for dialogue agents capable of goal-oriented behaviour [9] and, secondly, the technologies to develop them without the need to hand-craft the dialog flow are evolving very quickly in the last few years, bringing an active interest from the scientific community and, consequently, a strong proliferation of articles which could overwhelm anyone approaching the topic for the first time.

The research have been carried out on three of the most relevant digital libraries identified by Brereton *et al.* [10]: Google Scholar, ACM Digital and IEEExplore. I followed the standard practice of performing the automatic search within the titles, abstracts and keywords; no restrictions have been imposed on the year of publication. For each individual search, only the first 200 results were considered, ordered by relevance.

Search string: *("goal-oriented" OR "task-oriented") AND dialog* AND "end-to-end".*

Note: The wildcard in *dialog** has been used to include plurals and different notations often seen in literature such as *dialog* and *dialogue*.

Search results:

- Google Scholar, excluding patents and citations: 3780 results;
- ACM Digital Library: 95 results;
- IEEExplore: 12 results.

Finally, I've taken into account also new cited and citing articles when considered relevant to the study.

2.3 Selection criteria

In this phase I had to perform a first screening, considering both the relevance to the study goal and the scientific importance of the articles. First of all, basic rules for inclusion and exclusion have been set up.
Inclusions:

- Full articles, not PowerPoint presentations or extendend abstracts
- Papers published in peer-reviewed journals or conference proceedings
- Only sources about goal-oriented (or task-oriented) dialog systems, not about chit-chat bots
- Only sources with end-to-end trainable models

Exclusions:

- Reviews, which don't introduce new solutions
- Papers focused solely on performance evaluation of dialog systems
- Papers that introduces new datasets or new ways to retrieve data without proposing novel solutions
- Papers that were aimed at open-domain dialogue agents
- Papers related to Multi-modal or to Visual dialog
- Duplicates reports for the same study

Afterwards, since I wanted to include both the fundamental studies and the most promising ones based on them, I added papers prior to 2018 only if they had more than 20 citations, while most recent ones were included also with fewer or no citations, as long as they were accepted at conferences (or published in peer-reviewed journals). Using the described criteria, a total of 31 publications has been selected.

2.4 Synthesis

All the chosen studies are based on end-to-end trainable neural networks and, although the goal always remains to minimize human intervention, there are different levels of intervention necessary depending on the different solutions. Typically, a model's accuracy improves as manual tuning increases and, in general, as the designer makes the network capable of recognizing meaningful data types belonging to the domain of interest (see the "*Match type features*" in Memory Networks [1] or the "*Domain-specific action templates*" in Hybrid Code Networks [11]); consequently, it will be essential to always consider the best trade off between model accuracy and the need for human intervention, and possibly choose the most appropriate solution for the specific problem to solve.

I want to bring attention also to the different nature of the public datasets typically used in the studies: some of them just provide many different dialogues between the user and the conversational agent (e.g. Facebook's "*Dialog bAbI tasks*"), while others focus mainly on *explicitly* tracking the state of the conversation and the user's intentions, which must therefore be explicitly provided together with the dialogues (e.g. Microsoft's "*Dialog State Tracking Challenge*"). In general, although it has been shown that the ability of a conversational agent to track the state of the conversation is strongly linked to its effectiveness in the goal-oriented context, solutions capable of performing well even in the first kind of datasets will be considered preferable since, when applied to a real context, they allow the designer to avoid manual state and intent tagging.

Regarding the development of end-to-end solutions for goal-oriented agents, four fundamental deep learning frameworks were developed between 2016 and 2017, which many studies have then been based on and drew inspiration from in the following years, introducing new ideas and improvements:

- Deep Q-Networks
- End-to-End Memory Networks
- Sequence-to-Sequence Networks
- Hybrid Code Networks

The four reference frameworks are briefly discussed in the sub-chapters below, while the novel contributions of each of the most recent studies are directly reported in Table 2.1, alongside with the papers' titles, the frameworks on which they are based and the years of publication.

2.4.1 Base technologies for End-to-End Goal-Oriented Dialog Agents

Frameworks based on Deep Q-Networks

Since 2016 deep reinforcement learning solutions were proposed for dialog state tracking [12] and for information access via user interaction [13]. The former work proposed a *Recurrent* extension to Deep Q-Networks (DRQNs) which introduced an LSTM layer on top of the convolutional layer of the original DQN model, which allowed DRQN to solve POMDPs in dialogue settings. The major drawback of frameworks based on Deep Q-Networks is that they typically assume that the model has access to an explicit representation of the dialog state or, more in general, to a reward signal; such information could not be available in practice or would imply heavy manual intervention by the designer.

A notable work from 2018 tried to overcome this shortcoming by proposing an adversarial method to learn rewards directly from dialog samples [14]. Another study focused instead on a typical problem of dialog systems - knowledge base access: it proposed a deep Q-learning based system (Deep Q-network) capable of interact with a structured database to assist users in accessing information and accomplishing tasks; the reinforcement learning based dialog manager has been particularly able to handle noises caused by other components [15].

End-to-End Memory Networks (MemN2Ns)

MemN2Ns are a form of Memory Networks [16] which are trained end-to-end, and hence require significantly less supervision during training, making them more generally applicable in realistic settings [17]. Initially applied to question answering problems, MemN2Ns were subsequently exploited for goal-oriented conversational agents when Bordes *et al.* [1] proposed a test bed to break down the strengths and shortcomings of end-to-end dialog systems in goal-oriented settings: the *Dialog bAbI dataset*. Set in the context of restaurant reservation, its tasks required manipulating sentences and symbols, so as to properly conduct conversations, issue API calls and use the information provided by the outputs of such calls.

MemN2N-based studies relevance is twofold: first of all, since they only need corpora of dialogues as datasets there is no need to explicit the user's intentions nor the state of the conversation; secondly, because Memory Networks have been the starting point for many recent studies which aim to enhance dialog personalization based on user characteristics [18] [19], integration with external knowledge bases [20] [21], and modeling of long-range dialog history information [22].

Sequence-to-Sequence Networks (Seq2Seq)

Seq2seq is a family of machine learning approaches developed by Google for machine translation and then used in several natural language processing tasks. Seq2seqs are composed by an encoder and a decoder components and, in general, turn one sequence into another; they can make use of recurrent neural networks (RNNs) or, more often, LSTMs or GRUs. Having had extreme success for chit-chat systems, it's only natural that researchers would have tried to exploit Seq2Seq-based systems even in goal-oriented contexts. In 2017 neural encoder-decoder architectures has been used to frame goal-oriented dialogues as a sequence-to-sequence learning problem trainable both with reinforcement learning and supervised learning [23]. Furthermore, it's been added a copy mechanism while testing its effectiveness on the DTSC2 - a dataset for dialog state tracking.

In recent years, several studies have tried to further develop Seq2Seq networks applied to goal-oriented systems: Lei *et al.* [24] focused on reducing model complexity for better scalability; Hupkes *et al.* [25] investigated how encoder-decoder models process disfluencies, such as hesitations and self-corrections; Qin ed al. [26] proposed a novel framework which queries a knowledge base in two steps in order to improve the consistency of generated entities.

Hybrid Code Networks (HCNs)

HCNs are Recurrent Neural Networks (RNNs) based frameworks introduced in 2017 which augmented vanilla RNNs with two main components [11]: a *domain-specific knowledge encoded as software* and *system action templates*. The goal is to considerably reduce the amount of training data required while retaining the benefits of inferring a latent representation of dialog state and end-to-end trainability. The key advantage of this approach is the developer's ability to effectively inject domain knowledge and constraints.

Interestingly, although this type of framework requires some level of domain-specific human intervention, it reports state-of-the-art performance on the bAbI dialog dataset and even better performance than two commercial dialog agents [11]. Moreover, HCNs (like some Seq2Seq solutions) can be trained both with supervised and reinforcement learning. Two notable studies in particular have used HCNs: the first one demonstrated interactive teaching for end-to-end dialog control [27] while the other proposed a Hierarchical expansion (HHCNs) in order to conduct better semantic analysis and therefore select more meaningful responses to user's requests [28].

ID	Title	Framework	Novelties	Year
S01	Towards End-to-End Learning for Dialog State Tracking and Management using Deep Reinforcement Learning [12]	Deep Recurrent QNetworks (DRQN)	Implements a deep reinforcement learning based end-to-end framework for both dialog state tracking and dialog policy	2016
S02	Learning End-to-End Goal-Oriented Dialog [1]	End-to-End Memory Networks (MemN2N)	Proposes a new dataset for end-to-end goal-oriented dialog systems evaluation (Dialog bAbI) and adapts the MemN2N framework for goal-oriented tasks	2016
S03	Gated End-to-End Memory Networks [29]	Gated MemN2N	Exploits a gating mechanism in the context of End-to-End Memory Networks in order to regulate the access to the memory blocks in a differentiable fashion	2016
S04	End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning [30]	LSTM	Uses a recurrent neural network (an LSTM) which maps from raw dialog history directly to a distribution over system actions	2016
S05	A Network-based End-to-End Trainable Task-oriented Dialogue System [31]	Sequence-to-sequence networks (Seq2Seq) and RNN-CNN	Mixes POMDP and Seq2Seq approaches, having each module of the system end-to-end trainable	2016
S06	A Copy-Augmented Sequence-to-Sequence Architecture Gives Good Performance on Task-Oriented Dialogue [23]	Copy-Augmented Seq2Seq	Implements a recurrent neural dialogue architecture augmented with an attention-based copy mechanism	2017

ID	Title	Framework	Novelties	Year
S07	Hybrid Code Networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning [11]	Hybrid Code Networks (HCNs)	Combines an RNN with domain-specific knowledge encoded as software and system action templates	2017
S08	Demonstration of interactive teaching for end-to-end dialog control with hybrid code networks [27]	Hybrid Code Networks (HCNs)	A developer teaches the network by interacting with the system and providing on-the-spot corrections; once a system is deployed, mistakes can also be corrected from logged dialogs	2017
S09	End-to-end task-completion neural dialogue systems [15]	Deep Q-network (DQN)	Proposes a neural dialogue system that can directly interact with a structured database to assist users in accessing information and accomplishing certain tasks. The Reinforcement Learning based dialogue manager offers robust capabilities to handle noises caused by other components of the dialogue system	2017
S10	Iterative policy learning in end-to-end trainable task-oriented neural dialog models [32]	Deep Reinforcement Learning	Jointly optimizes the dialog agent and a user simulator with deep RL by simulating conversations between the two	2017
S11	End-to-end optimization of task-oriented dialogue model with deep reinforcement learning [33]	Hybrid supervised and deep Reinforcement Learning	The dialogue agent is trained in a supervised manner by learning directly from task-oriented dialogue corpora, and then optimized with deepRL during its interaction with users	2017

ID	Title	Framework	Novelties	Year
S12	Personalization in Goal-Oriented Dialog [18]	MemN2N with Split Memory architecture	Presents a new dataset of goal-oriented dialogs which are influenced by speaker profiles attached to them. Then analyzes the shortcomings of an existing end-to-end dialog system based on Memory Networks and propose modifications to the architecture which enable personalization	2017
S13	Subgoal discovery for hierarchical dialogue policy learning [34]	Subgoal Discovery Network (SDN) and Hierarchical RL	Divides a complex goal-oriented task into a set of simpler subgoals in an unsupervised fashion; then uses these subgoals to learn a multi-level policy by hierarchical reinforcement learning	2018
S14	Analysing the potential of seq-to-seq models for incremental interpretation in task-oriented dialogue [25]	Seq2Seq with attention mechanism	Concludes that recurrent networks with attention can learn to correctly process disfluencies, provided they were presented to them at training time; furthermore, suggests that the disfluencies contribute to a better understanding of the input, rather than hindering it	2018
S15	Hierarchical Hybrid Code Networks for Task-Oriented Dialogue [28]	Hierarchical Hybrid Code Networks (HHCNs)	a word-character RNN for semantic representation and a NN-based selection for domain knowledge are integrated	2018

ID	Title	Framework	Novelties	Year
S16	Bbq-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems [35]	Deep Q-learning	present a new algorithm that significantly improves the efficiency of exploration for deep Q-learning agents in dialogue systems	2018
S17	Dialogue learning with human teaching and feedback in end-to-end trainable task-oriented dialogue systems [36]	Hierarchical LSTM	Proposes a hybrid imitation and reinforcement learning method, with which a dialogue agent can effectively learn from its interaction with users by learning from human teaching and feedback	2018
S18	Sequicity: Simplifying Task-oriented Dialogue Systems with Single Sequence-to-Sequence Architectures [24]	Two Stage CopyNet, based on Seq2Seq	Proposes a novel, holistic, extendable framework based on a single sequence-to-sequence model which can be optimized with supervised or reinforcement learning	2018
S19	Goal-Oriented Chatbot Dialog Management Bootstrapping with Transfer Learning [37]	End-to-End Reinforcement Learning	Introduces a transfer learning method to mitigate the effects of the low in-domain data availability	2018
S20	Adversarial Learning of Task-Oriented Neural Dialog Models [14]	Generative adversarial networks (GANs)	Proposes an adversarial learning method to learn dialog rewards directly from dialog samples	2018
S21	Mem2Seq: Effectively Incorporating Knowledge Bases into End-to-End Task-Oriented Dialog Systems [20]	Mem2Seq	Proposes the first neural generative model that combines the multi-hop attention over memories with the idea of pointer network, aiming to better incorporate knowledge bases	2018

ID	Title	Framework	Novelties	Year
S22	Memory-to-Sequence learning with LSTM joint decoding for task-oriented dialogue systems [21]	Memory-to-Sequence: MemNN and LSTM joint decoding	Proposes a Memory-to-Sequence framework that uses Memory Neural Network (MemNN) and Long Short Term Memory (LSTM) joint decoding, in order to better capture the dependence between the system responses and the knowledge base items	2019
S23	A Modular Task-oriented Dialogue System Using a Neural Mixture-of-Experts [38]	Modular Task-oriented Dialogue System (MTDS)	A "chair bot" coordinates multiple expert bots and adaptively selects an expert bot to generate the appropriate response	2019
S24	End-to-End Question Answering Models for Goal-Oriented Dialog Learning [39]	Hierarchical RNNs, BiDAF, large-scale KB query methods from DrQA, Embeddings from Language Models (ELMo)	Uses popular approaches from both dialog and QA literature, and show that QA methods perform comparably well to the former, despite they were designed for a fairly different task	2019
S25	Incremental Learning from Scratch for Task-Oriented Dialogue Systems [40]	Incremental Dialogue System (IDS)	Introduces an uncertainty estimation to evaluate the confidence of giving correct responses; in case of low confidence humans will be involved in the dialogue process and the system can learn from human intervention. Also, a new dataset which simulates unanticipated user needs is provided	2019

ID	Title	Framework	Novelties	Year
S26	Memory-Augmented Dialogue Management for Task-Oriented Dialogue Systems [22]	MAD, a novel memory-augmented dialogue management	Employs a memory controller and two additional memory structures: the slot-value memory tracks the dialogue state by memorizing and updating the values of semantic slots, while the external memory augments the representation of hidden states of traditional RNN by storing more context information	2019
S27	Learning personalized end-to-end goal-oriented dialog [19]	Personalized MemN2N	Introduces a "profile model" which encodes user profiles into distributed embeddings and refers to conversation history from other similar users, then adds a "preference model" that captures user preferences over knowledge base entities, in order to better handle the ambiguity in user requests	2019
S28	An end-to-end goal-oriented dialog system with a generative natural language response generation [41]	FFNN with positional encoding	Generates the output word by word: bot responses are no longer restricted to a fixed number of candidates	2019
S29	Entity-Consistent End-to-end Task-Oriented Dialogue System with KB Retriever [26]	Seq2Seq with attention mechanism over a database	Proposes a novel framework which queries a Knowledge Base in two steps in order to improve the consistency of generated entities	2019

ID	Title	Framework	Novelties	Year
S30	DialogAct2Vec: Towards End-to-End Dialogue Agent by Multi-Task Representation Learning [42]	DialogAct2Vec	Proposes a novel joint end-to-end model by multi-task representation learning, which can capture the knowledge from heterogeneous information through automatically learning of knowledgeable low-dimensional embeddings from data	2019
S31	Hello, It's GPT-2 – How Can I Help You? Towards the Use of Pretrained Language Models for Task-Oriented Dialogue Systems [43]	GPT-2	Builds on top of the TransferTransfo framework and generative model pre-training, in order to validate the approach on complex multi-domain task-oriented dialogues from the MultiWOZ dataset	2019

Table 2.1: State of the art overview - for every selected study it's been reported a brief summary of the main novelties introduced, in addition to the reference framework and the year of publication.

Chapter 3

Working method

In agreement with my company supervisor we decided that, following the thesis goals, it would have made sense to invest about 85% of my time to study and develop the dialogue management system (see section 4.1.2), while in the remaining 15% I would have implemented the prototype's outline architecture (see section 4.2.2), including a bare bone client for the user interaction with an anthropomorphic virtual assistant.

3.1 Roadmap

Over the period between mid-November and the Christmas holidays I alternated the theoretical study of end-to-end goal-oriented dialogue systems with the development of the web-based client in which I embedded an anthropomorphic avatar with text-to-speech capabilities. Regarding the literature study, I described in detail the searching process, the selection criteria and the synthesis of my findings in Chapter 2.

From the post-Christmas period until the end of February I focused on the implementation of the core dialogue system, choosing the End-to-End Memory Networks [17] as reference framework, applied to goal-oriented dialog systems as proposed in Bordes *et al.* [1]. Together with my company supervisor, I've chosen this deep neural network architecture because it's considered extremely relevant in this research area, plus it's been the crucial starting point for several recent and promising studies [29] [18] [20] [19] [22] [21]. We kept each other regularly updated through weekly checkpoint, during which we reviewed my progresses, compared our findings and, finally, decided how to proceed until the next meeting on the basis of what had been done and learnt.

3.2 Obstacles and mindset

The two major obstacles I faced have been first of all trying to have a good grasp of the vast literature available from 2016 and, subsequently, learning how to use in a relatively short time tools for deep neural network development, with which I've never worked before. I often found myself alternating moments of study with sessions for practical implementation; during the first weeks it's been of great help being able to consult reference code examples and open source public repositories available on portals such as `github.com` and `paperswithcode.com`. It's been of great support and motivation also being able to regularly confront my company supervisor, exchanging ideas, articles and code snippets in order to find guidance.

Regarding the first obstacle, I constantly tried to reasonably balance the breadth and the depth of my studies, starting with a more superficial overview and then deepening the topics assessed as more relevant for my work. As for the second obstacle, I found extremely useful the availability of recent high-level frameworks for deep learning (e.g. Keras, see section 4.3.4) which proved themselves perfect for a beginner, thanks to their effectiveness, ease of use and community support.

During my time at IBM I also successfully took the last exams of my study plan: an undertaking made possible thanks to the freedom and organizational autonomy that the company gave me. I believe that this work experience, albeit relatively short, has helped me understanding how to better manage available time and resources while facing new challenges: a lesson that I will surely keep in mind for my future career.

Chapter 4

Thesis project

4.1 Scenario

The project goal is to develop a prototype which would allow a user to interact in spoken form with an anthropomorphic conversational agent, with the aim of receiving advice about risk and possible insurance plans based on the different kind of housing and heating systems. As already specified in the previous chapters, the conversational agent's model must be trained directly on dialogues; not being in possession of real conversations though, it will be necessary to generate them in a synthetic way. Inspired by the Dialog bAbi dataset for restaurant reservations, a corpora of dialogues must be created with the following structure:

1. First of all, following an initial greeting and the request from the user to receive information about risk, the conversational agent must effectively collect two information about the user: his/her type of dwelling and his/her kind of heating. Specifically, three possible types of housing (single house, multifamily, flat) and five types of heating (natural gas, LPG, wood, pellet and electric) are identified. Note that the conversational agent must be able to correctly handle different ways of collecting information, specifically the user could indeed consult it:

    - without immediately providing additional information, as seen in Figure 4.1, in which case the agent would have to proactively ask for both;
    - providing only one of the two, as seen in Figure 4.2 where the agent would have to ask for the other;

- immediately providing both, as seen in Figure 4.3, in which case the agent should identify them and immediately make the correct API call.

2. Once both information pieces have been collected, the agent must perform the correct API call to the knowledge base, in which relevant advice and insurance policies are stored; note that the agent must learn from the different conversations the various synonyms with which users could refer to a particular type of housing and heating. Of course, the same API call must be performed when users refer to the same thing differently; for example, "single house", "standalone house", "single family home" and "detached house" all refer to the same concept.

3. An API call will retrieve a couple of tips for risk prevention and a variable number - from one to three - of insurance solutions, with maximum, medium or minimum coverage. Of course such dialogue structure represents a simplification, but it's been considered an acceptable compromise, able to showcase some non-trivial skills to be learned with end-to-end training: in addition to the already discussed synonyms and provided information recognition, the conversational agent must learn to deal with different user behaviors, which can range from the most positive (see Figure 4.4) to the most adverse (see Figure 4.5).

    - First of all, the user is offered a first advice on risk prevention based on his home and heating information;
    - then he's asked if he is interested in a second advice and, if so, another tip is given;
    - subsequently, the user is asked if he'd like to learn more about a suitable insurance coverage;
    - in case of a positive response, the agent must offer the user the insurance with the highest coverage among those retrieved from the knowledge base with the API call.
    - if the user wants to know more about alternatives, the system must offer the next available insurance in descending order of coverage, until the user accepts, or until the available policies end, or until the user ends the conversation.

User	Conversational Agent
Good morning	
	Hi, what can I help you with today?
I'd like to know more about risk	
	What kind of housing do you live in?
A single house	
	What kind of heating do you use?
I use natural gas heating	
	Ok let me look into it
	api_call single natural_gas

Figure 4.1: Example of a conversation's preliminary phase where the user doesn't spontaneously provide any information. The conversational agent must therefore ask for both pieces of information before performing the API call.

User	Conversational Agent
Hello there	
	Hi, what can I help you with today?
Can you tell me what are the risks of living in a multi family house?	
	What kind of heating do you use?
I use lpg heating	
	Ok let me look into it
	api_call multi lpg

Figure 4.2: Example of a conversation's preliminary phase where the user spontaneously provide one piece of information. The conversational agent must therefore ask for the other before performing the API call.

User	Conversational Agent
Hi	
	Hi, what can I help you with today?
Can you tell me what are the risks of living in an apartment and using pellet heating?	
	Ok let me look into it
	api_call flat pellet

Figure 4.3: Example of a conversation's preliminary phase where the user spontaneously provide both pieces of information. The conversational agent must therefore immediately perform the correct API call.

User	Conversational Agent
	single_natural_gas_advice_1 do you want another advice?
Sure thank you	
	single_natural_gas_advice_2 do you want to see insurance plans?
Ok	
	What do you think of ***single_natural_gas_insurance_max***?
Do you have something else?	
	How about ***single_natural_gas_insurance_medium***?
Looks great to me	
	Great then let's proceed with ***single_natural_gas_insurance_medium***

Figure 4.4: Example of conversation continuation after having issued the API call. In this case the user accepts a second advice and shows interest in the second insurance coverage proposed by the agent.

User	Conversational Agent
	single_natural_gas_advice_1 do you want another advice?
No I'm good	
	Ok then, do you want to see insurance plans?
I'd rather not	
	Ok, have a nice day

Figure 4.5: Example of conversation continuation after having issued the API call. In this case the user is particularly adverse since he rejects both a second advice and insurance plans information.

4.2 Logical architecture

4.2.1 Project plan

The project's most interesting part lies in the *Dialog Management System*, described in detail in section 4.2.2; all the other components are less relevant, but developed nonetheless in order to deploy a complete prototype, which can better showcase the project results and eventually be used as a starting point for future works.
Five main components are identified as shown in Figure 4.6:

- The central component must both provide a simple user interface equipped with an *Embodied Conversational Agent*, and act as an orchestrator with the other components, actively requesting Speech-to-Text and Text-to-Speech translations, asking the *Dialog Management System* for the proper dialogue responses, and finally querying the *Domain-specific Knowledge Base*.
- As already mentioned, most of the development time must be spent on the Dialog Management System, being the most relevant component for the thesis' purpose. It will embed the model trained on corpora of dialogs and will offer a way to ask for the proper conversational agent response by providing the last user utterance and the conversation history.
- An external Speech-to-Text service will be used to convert the sentences spoken by the user into textual format.
- For the translation of the answers provided by the Dialog Management System into speech format, it will be used another external service which will also provide an embodied conversational agent that will animate properly while pronouncing the sentences.
- Finally, it must be present a Knowledge Base storing domain-specific information; in our case study it would incorporate advice for risk prevention and proposals for different insurance policies, organized by type of housings and heating systems.

It's possible to consult Figures 4.7 and 4.8 to better understand the flow of interactions happening between the different components, from the user request until the system response.

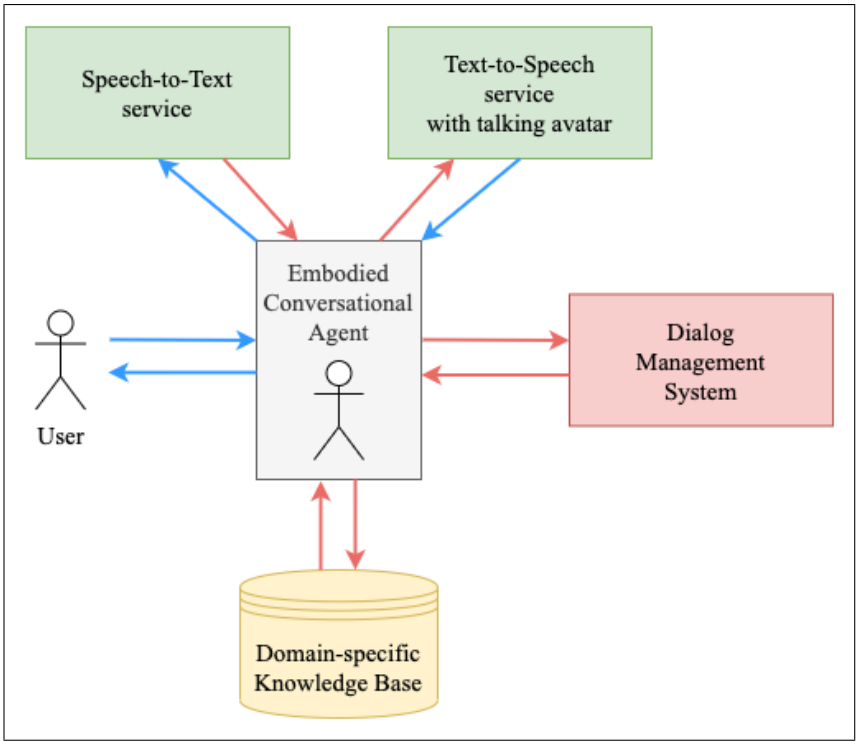

Figure 4.6: **Prototype's architecture overview**. Blue arrows represent exchange of audio information (speech) while red ones are for textual data. The yellow container incorporates information about risk hazards for different domestic heating solutions; the red box contains the most important part of the thesis project: the prediction model for the conversational agent's answers; the green boxes represent services for text-speech conversion and avatar visualization; finally, the gray box represents a component which deals with user interaction and system orchestration.

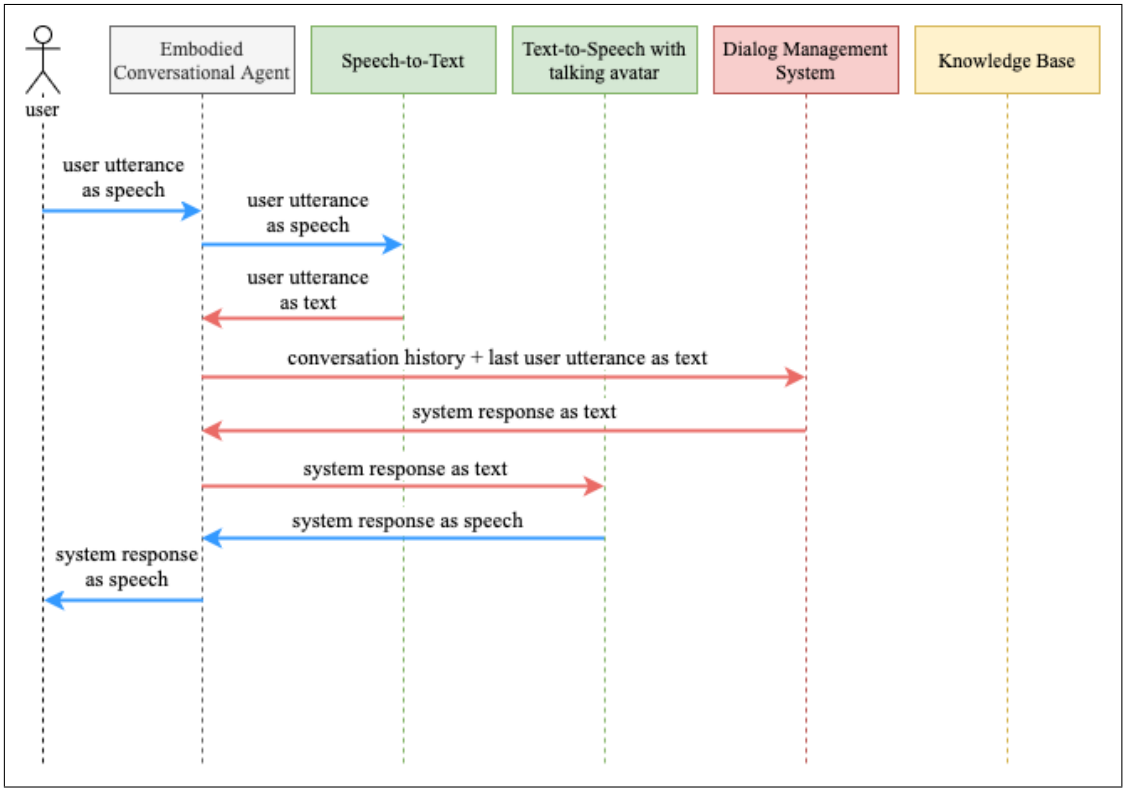

Figure 4.7: The diagram shows what kind of data is exchanged between which system components between a user request and the system response. After the user pronounces an utterance, a transcription service is exploited to obtain its textual representation. The transcription is then provided as input to the Dialog Management System, together with all the previous dialogue sentences. A proper response is evaluated and sent to a Text-to-Speech translation service, which would also provide proper talking avatar animation.

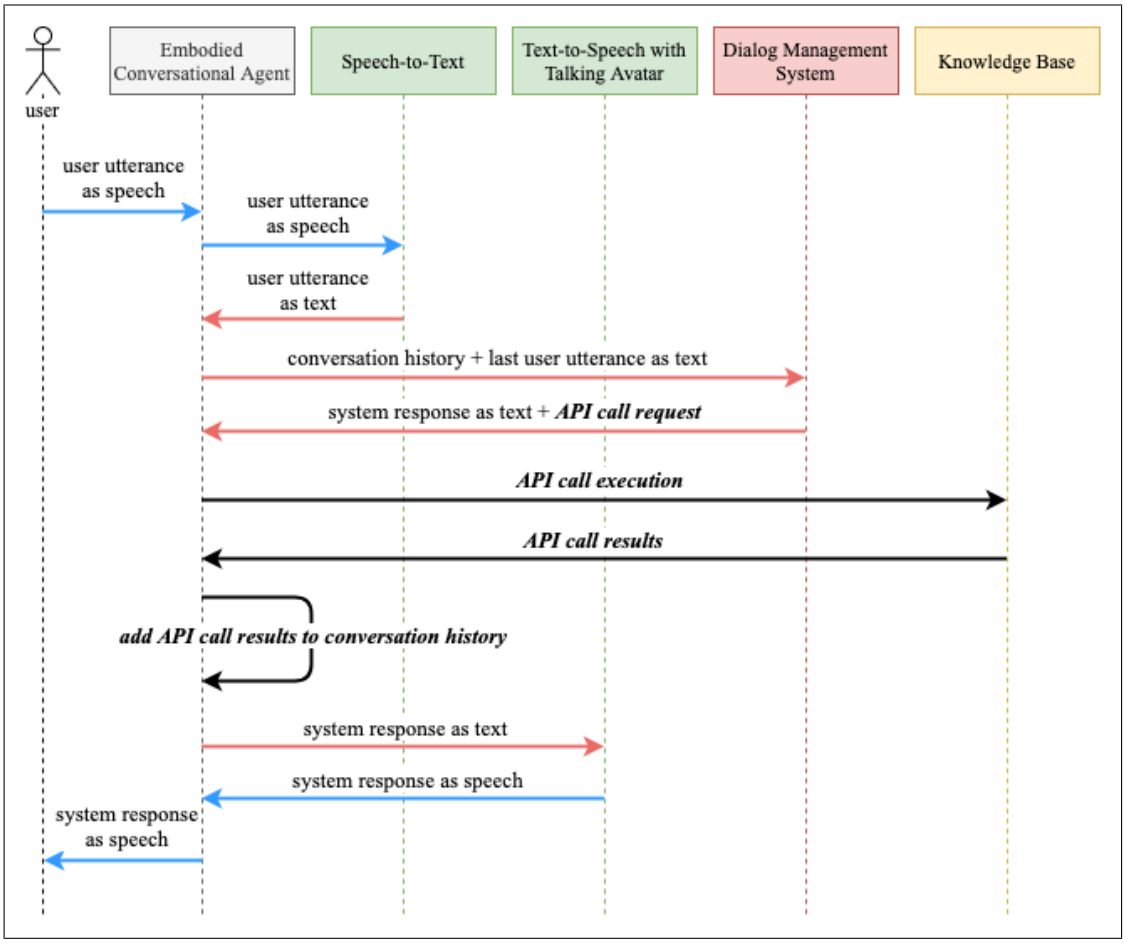

Figure 4.8: The diagram shows the interaction flow between the system components as already described for the Figure 4.7, with the addition of an API call request by the Dialog Management System. The orchestrator takes care of it by retrieving the requested information from the Knowledge Base and, subsequently, adding it to the conversation history; doing so, the obtained knowledge will be available in memory for the rest of the conversation.

4.2.2 Dialog Management System

The Dialog Management System is modeled exploiting the End-to-End Memory Network framework (*MemN2N*) proposed by Facebook AI Research in 2015: a neural network with a recurrent attention model over an external memory, see Figure 4.9, initially applied to NLP tasks such as question answering and language modeling [17]. In 2016 Bordes *et al.* [1] used MemN2N - making some small changes to the baseline - for goal-oriented dialog systems, testing it with a dataset of conversations for restaurant reservations created ad hoc: the *Dialog bAbI dataset*.

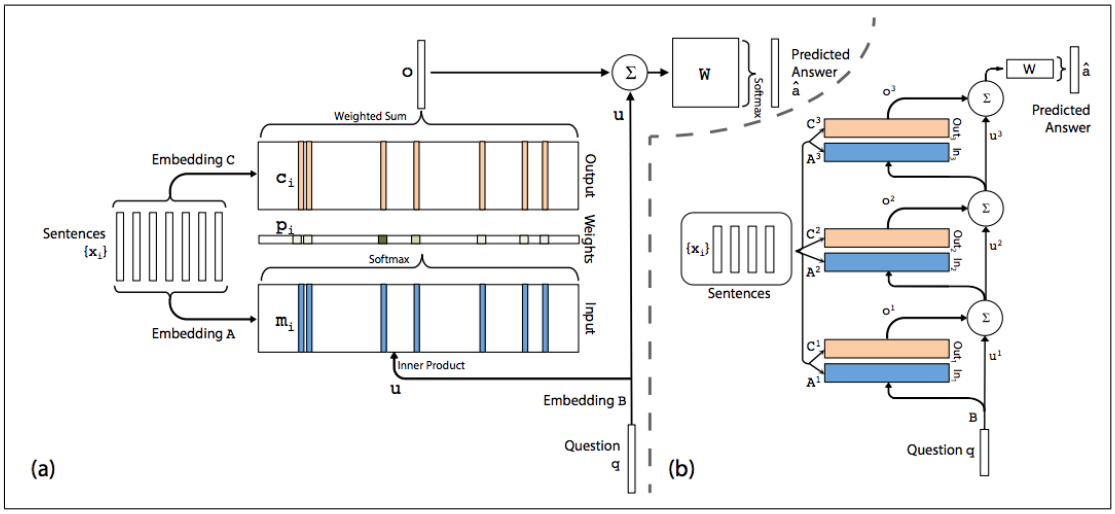

Figure 4.9: Single (a) and triple (b) layer version of the End-to-End Memory Network model proposed by Sukhbaatar *et al* [17].

Model description with an example

The key concepts of the model are (i) how it stores the conversation in memory, (ii) how it reads from memory to reason about the proper response, and (iii) how it outputs the response. In order to better understand the model explanation I'll make use of the simple example shown in Figure 4.10, where the user and the conversational agent are in the middle of a brief conversation.

(i) As the model conducts a conversation with the user, at each time step t the previous user utterance and model response are appended to the memory. Therefore, at any given time there are $c_1^u, \dots c_{t-1}^u$ user utterances and $c_1^r, \dots c_{t-1}^r$ model responses stored in memory (the entire conversation). The goal at the time t is, given the conversation and last user utterance c_t^u, to choose the appropriate response c_t^r. Taking as a reference the example

in Figure 4.10 we'd have c_1^u and c_1^r in memory, c_2^u as the last user's utterance, c_2^r as the response to predict, where:

$$t = 2$$

$$c_1^u = \text{“}Good\ morning''$$

$$c_1^r = \text{“}Hi\ what\ can\ I\ help\ you\ with\ today?''$$

$$c_2^u = \text{“}I'd\ like\ to\ know\ more\ about\ risk''$$

$$c_2^r = \text{“}What\ kind\ of\ housing\ do\ you\ live\ in?''$$

Every utterance of the dialog is encoded as a bag of words of dimension $V = (V^* + T + 2)$ by the $\Phi(\cdot)$ function, where V is the extended vocabulary size, V^* is the original vocabulary size (which counts all the possible words present in the dialog corpora), T is equal to the maximum number of turns that we estimate could happen during a conversation, and finally the other two dimensions are added in order to specify when an utterance has been said by the user and when by the conversational agent. Following the example as in Figure 4.11 we'd have:

$$V = 602$$

$$V^* = 500$$

$$T = 100$$

After having listed in alphabetical order all the possible V^* words contained within the dialogues and having assigned them to a positional index, for every utterance c the resulting bag of words vector $\Phi(c)$ is composed by three parts:

- the first $V^* = 500$ elements consist of all zeroes except for the ones at the same positional index of words occurring in the utterance (in Figure 4.11 the only elements equals to 1 for the utterance c_1^u are the ones at position 84 and 204, corresponding to the words *good* and *morning* respectively);
- the following $T = 100$ elements are all set to zero except for the one corresponding to the current turn, which is set to 1;
- finally, the last two elements are always equals to 1 and 0, or to 0 and 1, depending if the utterance has being said by the user or by the conversational agent.

Every past utterance encoded as bag of words is then embedded with a matrix A of dimension $d \times V$, where d is the embedding size; these embedded utterances compose the network's memory component "m":

$$m = \left(A\Phi\left(c_1^u\right), A\Phi\left(c_1^r\right) \ldots, A\Phi\left(c_{t-1}^u\right), A\Phi\left(c_{t-1}^r\right)\right)$$

(ii) During training, the system must learn how to *reason over the memory* in order to identify which one of the past utterances could be relevant in choosing the right response; let's see how this translate mathematically. The last user utterance c_t^u is also embedded using the matrix A, giving $q = A\Phi\left(c_t^u\right)$, which is called "*the controller state*". As shown in Figure 4.12, the *match* between q and the memory - which indicates which are the most relevant memories - is computed by taking the inner product followed by a softmax: $p_i = \text{Softmax}\left(u^\top m_i\right)$, giving a probability vector over the memories. These probabilities are then multiplied for the corresponding embedded memories, and finally combined as follow in order to compute $o = R\sum_i p_i m_i$ where R is a $d \times d$ square matrix. The controller state is therefore updated with $q_2 = o + q$.

Note that the memory can be iteratively reread in order to refine the choice of relevant past utterances using the updated state q_2 instead of q, and in general using q_h on iteration h, with a fixed number of N iterations (called *N-hops*). The Figure 4.13 shows a *3-hops* representation of the model.

(iii) The final prediction is defined as:

$$\hat{a} = \text{Softmax}\left(q_{N+1}^\top W\Phi\left(y_1\right), \ldots, q_{N+1}^\top W\Phi\left(y_C\right)\right)$$

where y_i are all the candidates responses with i ranging from 1 to C, and W is a weight matrix of dimension $d \times V$. The candidates are composed by all the possible conversational agent's responses, including the API calls.

The entire model (the weights of the matrices A, R and W) is trained using stochastic gradient descent, minimizing a standard cross entropy loss between $\hat{a}$ and the true label a.

User	Conversational Agent
Good morning	
	Hi what can I help you with today?
I'd like to know more about risk	
	What kind of housing do you live in?

Figure 4.10: A preliminary exchange between the user and the conversational agent, used as a simple example for the model description: the utterances c_1^u="*Good morning*" and c_1^r="*Hi what can I help you with today?*" are part of the conversation history (stored in memory); the utterance c_2^u="*I'd like to know more about risk*" is the last user's utterance and c_2^r="*What kind of housing do you live in?*" is the response we want to predict.

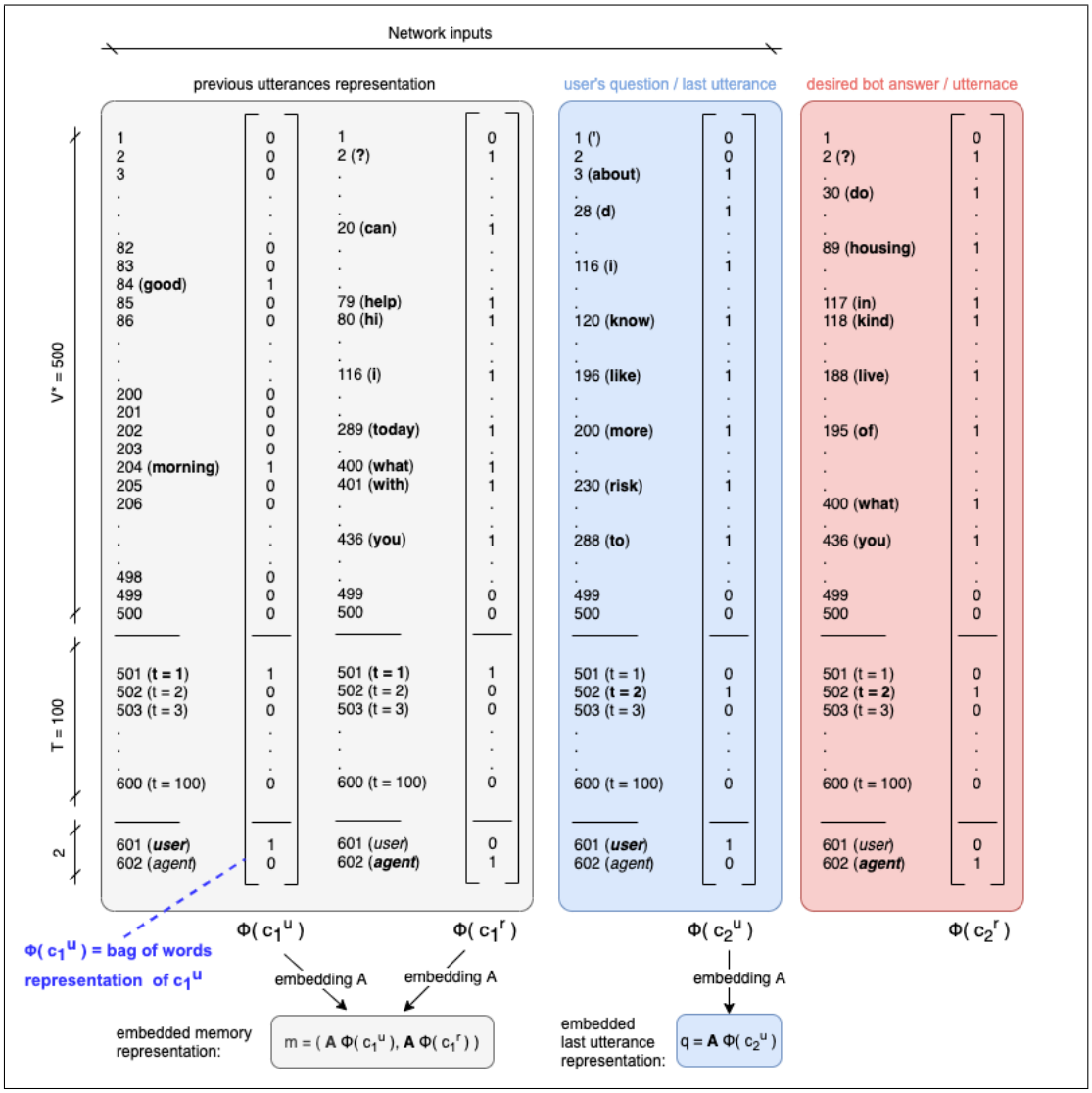

Figure 4.11: **Bag of words representations of the four utterances showed in the dialog of Figure 4.10**. In this example we assumed a vocabulary dimension V of 500 words, extended by T=100 time features and two further features which indicate if an utterance was said by the user or by the conversational agent. For each sentence the words contained inside it are shown on the left, alongside their numerical index, increasing in alphabetical order from 1 to 500 (the size of V); on the right side it's reported the corresponding vectorial representation as bag of words. All the values non explicitly shown are considered equal to zero.

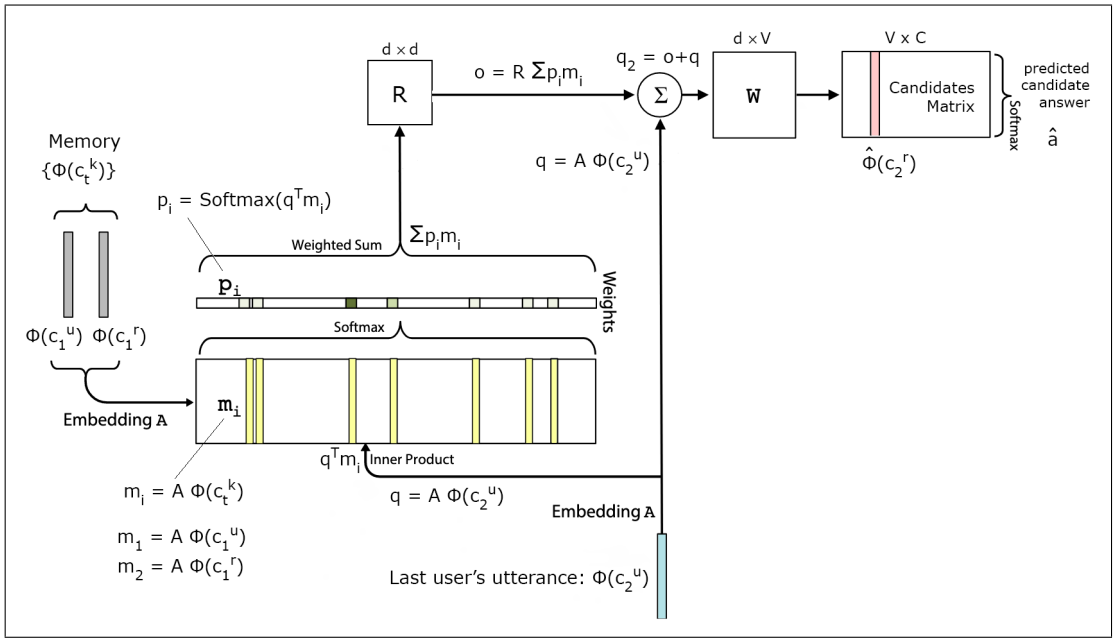

Figure 4.12: **Mathematical model of the End-to-End Memory Network (MemN2N) used for Goal-Oriented dialog systems as described in Bordes *et al*** [1]. I've drawn this picture taking inspiration from the one shown in Sukhbaatar *et al* [17], see Figure 4.9, bringing some minor changes described in [1]. Note that, for a better understanding, light blue, gray and red rectangles represent utterances as bags of words with the same color code used in Figure 4.11. Yellow rectangles represent the memories embedded with the A matrix, while $\mathsf{p_i}$ are drawn with different green intensity in order to indicate different probability values that they can assume. Note that there's only one kind of embedding for the memories (in yellow), opposed to the two embeddings present in figure 4.9 (in blue and salmon color): this derives from the imposed constraint on the two embedding matrices $\mathsf{A} = \mathsf{C}$, as suggested in Bordes *et al* [1]. The same A matrix is also used for the embedding of the last user's utterances because of the constraint $\mathsf{A} = \mathsf{B}$, applied to limit the neural network complexity.

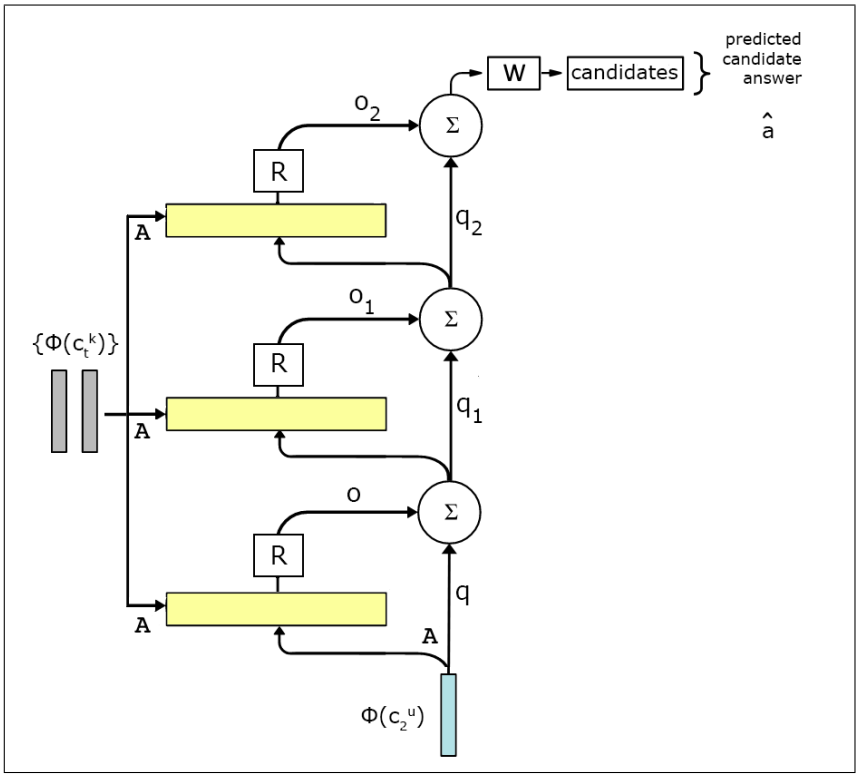

Figure 4.13: **Multi-hop model of the End-to-End Memory Network**. It shows how to compose a 3-hops neural network; for a detailed view of the single hops refer to the Figure 4.12. I used the *layer-wise* weight tying method described in [17], with which the embeddings are the same across different layers, i.e. $\mathsf{A}^1 = \mathsf{A}^2 = \mathsf{A}^3 = \mathsf{A}$.

API calls mechanism

Three steps are followed in order to integrate external knowledge into the dialogs:

1. when the conversational agent has collected enough information to query the knowledge base (e.g. the user's type of housing and heating system), it must respond with coded sentences which indicate an API call request: "*api_call single pellet*" can be used to request domain knowledge about pellet heating systems for users living in single houses;

2. a software component interposed between the conversational model and the user must be able to recognize these requests and, instead of reporting them to the user, it must perform the proper information retrieval on the database;

3. finally, once the responses from the external database are received, the software component

must inject them into the current dialog by adding them to the conversation memories; this way, the Dialog Management System will be able to reason about them in the subsequent interactions with the user.

If it's necessary for the system to correctly interpret information present in the knowledge base but absent in the training set, it would be possible to exploit the *match type feature* described in Bordes *et al.* [1], as long as the new information is of the same "type" of other already learnt by the model during the training phase.

4.3 Implementation and Technologies

Regarding the PoC implementation, I kept in mind the order of priorities assigned by my company supervisor: I devoted most of my time and efforts to the development of the Dialog Management System. His indication has also been reflected in the technological choices: for example, the handy HTML5 Web Speech API has been considered a quick and reasonable compromise although still not supported by some browsers (see Section 4.3.2), while I spent some time to better document, refactor, and optimize the Python code of the neural network model.

In Figure 4.14 it's reported an overview of the main technologies adopted for the prototype's development. They're broken down below in the following sections: Front end, Speech-to-text service, Text-to-Speech with talking avatar, Dialog Management System, Dialog corpora generation.

4.3.1 Front end

Given the prototypical nature of the project and the versatility of Web based solutions, I decided to develop the front end as a simple Web page using HTML, CSS and plain JavaScript. The result is shown in Figure 4.15: the talking avatar was chosen from the BotLibre's free catalog and it proved itself adequately flexible and easy to use thanks to the provided JavaScript's Web SDK; under the virtual agent I've inserted three buttons and two text areas. By pressing the first button - push to talk - it's possible to speak and see the live transcription inside the first text area; doing so, if the user wants to change the text transcription, he could do it before confirming with the send button. Conversely, inside the disabled text area at the bottom it's listed the conversation history, which can be reset by pressing the the button clear history.

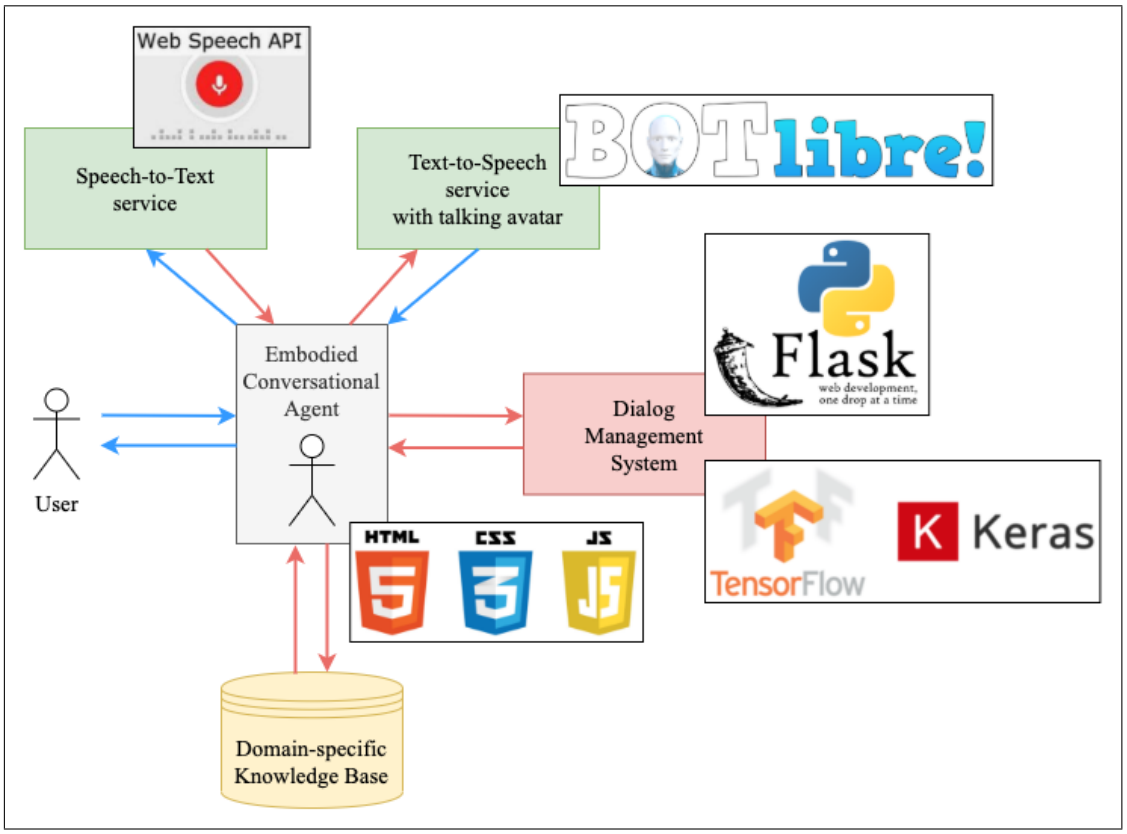

Figure 4.14: Prototype's implementation technologies applied on the system architecture overview previously shown in Figure 4.6.

In addition to managing user interaction, the front end also deals with:

- Asking the Dialog Management System for the agent's response, issuing a POST request every time the user press the `send` button. The request body is a JSON file which include the conversation history and the last user utterance, see Figure 4.16 and 4.17 for more details.
- Questioning the Knowledge Base when a response from the Dialog Management System contains an API call, and then appending the results inside the `context` field in the future POST requests, see Figure 4.17. The Knowledge Base has been implemented as a set of textual files, which contain hypothetical information about risk prevention and insurances for different housing and heating systems.

4.3.2 Speech-to-text

The HTML5 Web Speech API aims to grant an alternative input method for web applications and provides two distinct areas of functionality: speech recognition (with the *Speech Input API*), and speech synthesis (with the *Text to Speech API*). I only made use of the former because, for the speech synthesis, I exploited the text-to-speech provided by BotLibre - see Section 4.3.3. The API itself is agnostic of the underlying speech recognition implementation and can support both server based as well as embedded recognizers. For example, Chrome implementation of speech recognition involves Google's server-based recognition engine: the audio is sent to a Web service for recognition processing, so it won't work offline.

Support for Web Speech API speech recognition is currently limited to Chrome for Desktop and Android, but it can also be enabled in recent versions of Firefox Nightly. Given the experimental nature of the project and the convenience of these new APIs, I decided that despite the current limited support, using HTML5 Web Speech API would have been a good compromise. In case the system prototype should be further developed in the future - and several browsers won't have yet implemented the API - it will certainly be necessary to make use of alternative services to obtain better browser coverage, such as IBM Speech to Text, Microsoft Bing Voice Recognition, Google Cloud Speech API, Cedat85, Wit.ai, Houndify API, CMU Sphinx, etc.

Figure 4.15: Web-based user interface with an anthropomorphic conversational agent.

```
{
    "context": [
        ["1", "u", "good morning"],
        ["1", "r", "hello what can i help you with today"]
    ],
    "utterance": "I'd like to know more about risk"
}
```

Figure 4.16: **Simple example of JSON document used in the POST request body when asking the Dialog Management System's Web server for the correct response**. The `context` field is a list of lists containing the conversation's past utterances (the memories); every sub-list is composed by three strings, where the first one represent the turn, the second indicates if it's a user utterance "`u`" or a conversational agent response "`r`", and the third one is the actual sentence. The `utterance` field represent instead the last user utterance, to which the Dialog Management System must respond.

4.3.3 Text-to-speech with talking avatar

As a requirement, it was mandatory to embed an anthropomorphic virtual agent into the application's UI, since these kind of avatars can add something to the interaction that, for us humans, is viscerally different when perceived as a face-to-face conversation. Not being able to invest in professional, paid solutions, I found an excellent alternative - the BotLibre framework - which describe itself on `botlibre.org` as an "*Open source chatbot and artificial intelligence platform*".

Note that this platform allows users to create their own chatbot from scratch, or simply make use of many graphic and voice assets (3D models, animations and voices for speech synthesis). Of course I only made use of the provided cosmetic and audio assets (see the model I chose in Figure 4.15) because I implemented the conversational model's logic in the Dialog Management System; however, for information purposes I want to report how BotLibre also allows novices to experiment and create their own assistant with no programming skills, following the tutorials and examples shown in the website's forum section. I used the SDK for JavaScript, but BotLibre also provides development kits for Java, Android and iOS. My experience with the framework has been generally positive: it's very quick to set up and to use, even if not always well documented or refined.

```
{
    "context": [
        ["1", "u", "good morning"],
        ["1", "r", "hello what can i help you with today"],
        ["2", "u", "i'd like to know more about risk while living in a single house"],
        ["2", "r", "what kind of heating do you use"],
        ["3", "u", "i use pellet heating"],
        ["3", "r", "ok let me look into it"],
        ["4", "u", "<SILENCE>"],
        ["5", "r", "api_call single pellet"],
        ["6", "r", "<ADVICE1> single_pellet_advice_1 <housing_type> single"],
        ["7", "r", "<ADVICE1> single_pellet_advice_1 <heating_type> pellet"],
        ["8", "r", "<ADVICE2> single_pellet_advice_2 <housing_type> single"],
        ["9", "r", "<ADVICE2> single_pellet_advice_2 <heating_type> pellet"],
        ["10", "r", "single_pellet_insurance_medium <housing_type> single"],
        ["11", "r", "single_pellet_insurance_medium <heating_type> pellet"],
        ["12", "r", "single_pellet_insurance_medium <coverage_type> medium"],
        ["13", "r", "single_pellet_insurance_min <housing_type> single"],
        ["14", "r", "single_pellet_insurance_min <heating_type> pellet"],
        ["15", "r", "single_pellet_insurance_min <coverage_type> min"],
        ["16", "u", "<SILENCE>"],
        ["16", "r", "single_pellet_advice_1 - do you want another advice?"],
        ["17", "u", "nope"],
        ["17", "r", "ok then do you want to see insurance plans?"],
        ["18", "u", "yes please"],
        ["18", "r", "what do you think of single_pellet_insurance_medium?"]

    ],
    "utterance": "do you have an alternative"
}
```


Figure 4.17: **Another example of JSON document used in the POST request body when asking the Dialog Management System for the correct response**. I reported this example more complex than the one in Figure 4.16 to illustrate how the results of an API call are encoded in memory: the individual results retrieved from the Knowledge Base are provided within the context of the conversation, encoded as if they were dialog system's responses.

```python
class EmbeddingMatrixA(Layer):
    """
    Embedding matrix A of size dxV, where:
    - d is the embedding size
    - V is the vocabulary size
    It's used to embed both the user utterance and the facts inside the memory (the context).
    """

    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(EmbeddingMatrixA, self).__init__(**kwargs)

    def build(self, input_shape):

        # Note: The vocab size will be the last dimension for both the kind of tensors embedded with A:
        #       - the memories tensor will have the shape: (batch size, memory size, vocab size)
        #       - the user's utterance tensor will have the shape: (batch size, vocab size)
        vocab_size = input_shape[-1]

        self.A = self.add_weight(name='A',
                                 shape=(vocab_size, self.output_dim),
                                 initializer='random_normal',
                                 trainable=True)
        super(EmbeddingMatrixA, self).build(input_shape)

    def call(self, input, mask=None):
        return K.dot(input, self.A)
```

Figure 4.18: Python code snippet of the embedding matrix A.

4.3.4 Dialog Management System

The core component of the Dialog Management System consists of the dialog model which, given the last user utterance and the conversation history, returns the agent's response. It's been developed using Keras' high-level neural network APIs with TensorFlow back end; I have shown in Figures 4.18, 4.19, 4.20 the code snippets relative to three sections of the neural network and, in Figure 4.21, how they are composed in order to obtain the final model.

In order to make it possible to query the dialog model from external components - in our case from the Web client - I instantiated a simple Web server using the Flask Web application framework for Python: each time the server receives an appropriate POST request at the */predict* address it appropriately vectorize the utterances passed in the request body, and then consults the pre-trained dialog model (see examples of JSON request bodies in Figures 4.16 and 4.17).

Finally, I want to point out some optimization measures I made, which I found useful during the dialog model training phase: in terms of time, using a simple caching system to save the already vectorized dialogues in a *pickle* file; in terms of space representing the bag of words as *uint8* rather than the default *float32*.

```python
class MatchingSection(Layer):
    """
    Neural Network section that matches the embedded question with the embedded memories.
    """

    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(MatchingSection, self).__init__(**kwargs)

    def build(self, input_shape):
        embedded_memories_shape = input_shape[0]  # (batch size, memory size, embedding size)
        embedded_question_shape = input_shape[1]  # (batch size, embedding size)
        embedding_size = embedded_memories_shape[2]
        assert (embedding_size == embedded_question_shape[1] == self.output_dim)

        # Matrix weights of size dxd
        self.R = self.add_weight(name='R',
                                 shape=(embedding_size, embedding_size),
                                 initializer='random_normal',
                                 trainable=True)

        super(MatchingSection, self).build(input_shape)

    def call(self, inputs, mask=None):
        # Tip: Print K.int_shape(your_tensor) to see its dimensions:
        #      don't know why, but some debuggers have problems in this code section.

        embedded_memories = inputs[0]
        embedded_user_utterance = inputs[1]

        inner_product = K.batch_dot(embedded_memories, embedded_user_utterance)
        weights = K.softmax(inner_product)
        weighted_embedded_memories = K.batch_dot(weights, embedded_memories)

        o = K.dot(weighted_embedded_memories, self.R)

        return embedded_user_utterance + o
```

Figure 4.19: **Python code snippet of the matching section**. This portion of neural network implementation is used to identify which memories are most relevant in choosing the candidate response. Within the call() function it's possible to see a dot product between the embedded memories and the last user utterance, followed by a softmax in order to evaluate which weights to attribute to the different memories, and then to multiply these weights with the embedded memories. Another dot product is made between the previous result and the R matrix, which result is finally added to the embedded user utterance. Note that this section will be called N times depending on the number of *hops* to perform on the memory, providing at each iteration the result of the previous call as the new embedded_user_utterance. Refer to the *for* loop shown in Figure 4.21 for the details.

```python
class FinalSection(Layer):
    """
    Neural Network section used to choose the final response, it:
        - multiplies the input for a W matrix of size dxV, then
        - multiplies it for the candidates matrix (with all the candidates encoded as bags of words), and finally
        - applies a softmax.
    """

    def __init__(self, output_dim, candidates_matrix, **kwargs):
        self.output_dim = output_dim
        self.candidates_matrix = candidates_matrix
        super(FinalSection, self).__init__(**kwargs)

    def build(self, input_shape):
        input_dim = input_shape[1]  # remember that input_shape[0] is the batch size
        self.W = self.add_weight(name='W',
                                 shape=(input_dim, self.candidates_matrix.shape[0]),
                                 initializer='random_normal',
                                 trainable=True)

        super(FinalSection, self).build(input_shape)

    def call(self, input, mask=None):
        w_output = K.dot(input, self.W)
        matched_candidates = K.dot(w_output, self.candidates_matrix)
        return K.softmax(matched_candidates)
```

Figure 4.20: **Python code snippet of the final neural network section**. This snipped implements the portion of the model used for selecting the dialogue system response amongst all possible candidates. Within the call() function it's possible to see the dot product between the *input* - the matching section's output of Figure 4.19 - and the W matrix, followed by another dot product with the candidates matrix and, finally, a softmax. Note that the candidates matrix is composed by all the possible responses represented as bag of words and stacked side by side.

```
    vocabulary_size = params.candidates_matrix.shape[0]
    candidates_size = params.candidates_matrix.shape[1]

    memories_tensor = Input(name='memories_tensor',
                            shape=(params.context_size,  # the memory size: the max number of facts we need to remember
                                   vocabulary_size))     # the vocabulary size, including all the words for every task
    memories_tensor = K.cast_to_floatx(memories_tensor)

    utterance_tensor = Input(name='user_utterance_tensor',
                             shape=(vocabulary_size, ))
    utterance_tensor = K.cast_to_floatx(utterance_tensor)

    A = EmbeddingMatrixA(output_dim=params.embedding_size)  # 'A' shape: (embedding size, vocabulary size)
    embedded_m = A(memories_tensor)                         # embedded_m shape: (batch size, memory size, embedding size)
    embedded_u = A(utterance_tensor)                        # embedded_q shape: (batch size, embedding size)

    for _ in range(params.hops):
        embedded_u = MatchingSection(output_dim=params.embedding_size)([embedded_m, embedded_u])

    answer_tensor = FinalSection(output_dim=candidates_size,
                                 candidates_matrix=K.cast_to_floatx(params.candidates_matrix)
                                 )(embedded_u)

    return Model(inputs=[memories_tensor, utterance_tensor], outputs=answer_tensor)
```

Figure 4.21: **Python code snippet of the neural network composition**. Within this piece of code it's illustrated how the sections defined in Figure 4.18, 4.19 and 4.20 have been used to compose the complete neural network. First of all, both the user's utterance tensor and the memories tensor are embedded with the A matrix. After that, the *matching section* is repeatedly invoked according to the number of hops; the first time it will take the embedded user utterance as input, while the following invocations will take the result of the previous invocation as the new input. The predicted candidate index will be finally computed as the output of the FinalSection.

```
HOUSINGS = {
    "single": ("single house", "standalone house", "single family home", "detached house"),
    "multi": ("multifamily", "multiple family house", "multi family house"),
    "flat": ("apartment", "flat")
}
HEATINGS = {
    "natural_gas": ("natural gas heating", "heating with natural gas"),
    "lpg": ("lpg heating", "lpg"),
    "wood": ("wood heating", "heating with wood"),
    "pellet": ("pellet heating", "heating on pellets", "pellet heating system", "pellet firing"),
    "electric": ("electric heater", "space heater", "small electric heater")
}
```

Figure 4.22: **Synonyms considered during dialog generation for housings and heating systems**. During the training, the dialogue system must be able to implicitly connect the synonyms to the same concept. HOUSINGS is a Python dictionary where the keys correspond to the string to use in the API calls, while the values are tuples containing many ways in which the user could refer to the related concept. Same concept applies to the dictionary HEATINGS, which describes the various types of heating systems and the various ways in which they can be referred to.

4.3.5 Dialog corpora generation

Not having access to real dialogues, I had to synthetically generate a set of simulated conversations. The dialog generation turned out to be a critical and educational exercise, as it led me to a better understanding of the model's potential and limitations while I tried to correctly balance the diversity and the complexity of the dialogues I was generating. I took the Dialog bAbI dataset for restaurant reservations as base reference, trying to adapt the challenges and peculiarities proposed in it to the different context of risk awareness as described in Section 4.1. Doing so, I better understood some of the choices made by the researchers who generated the Dialog bAbI dataset, particularly regarding the interaction with the knowledge base and the proposal of alternative restaurants which should take place in a specific order.

For each type of dwelling and heating system I have identified various synonyms (see Figure 4.22), and used them to generate different combinations of dialogues. I did the same for the different ways in which the user could confirm or deny proposals from the conversational agent (see Figure 4.23).

```
USER_CONFIRMATIONS = ("yes", "yeah", "yes sure", "yes please", "sure", "of course", "ok", "i'd love to")
USER_NEGATIONS = ("no", "nah", "i'd rather not", "no thank you", "no thanks", "nope", "no i'm good", "no i'm fine")
USER_ACCEPTANCES = (
    "let's do it",
    "seems good to me",
    "that looks great",
    "seems perfect for my needs",
    "sounds good to me")
USER_REJECTIONS = (
    "do you have something else",
    "how about something else",
    "do you have another plan",
    "do you have an alternative",
    "are there other options"
)
```

Figure 4.23: **Synonyms for user confirmations, negations, acceptances, rejections used in dialog generation.** The different synonyms are used to generate a set of conversations with a certain variety, and to help the dialogue system to learn typical ways with which the user could answer yes/no questions and proposals.

4.4 Licensing

In the solution I implemented, the only licensed component is the BotLibre anthropomorphic avatar, which uses the first version of the *Eclipse Public License* (EPL1). It's a license aimed at commercial use, in particular I must only remember that: I need to mention that my project includes EPL code, and allow third parties to request access to the source code of the EPL section if they want, including any modifications I've made to it. My own code though, which only uses the EPL code in an import, does not need to be made EPL1.

4.5 Limitations

Since the End-to-End training of Goal-Oriented conversational agents is an extremely open and recent research area, I was already aware that the final prototype would inevitably present several compromises and limitations. I list below the most relevant ones:

- The system is unable to handle any word outside of the (limited) dictionary seen during the training phase.
- The conversational agent's response is chosen, at each turn, from a list of candidates (all the possible answers learned during the training); more sophisticated solutions use a generative approach instead.

- The prototype should be tested on a real dataset, since more or less satisfactory results currently strongly depend on the complexity with which the synthetic dialogues are generated.

4.6 Future development

During the last months I often found myself faced with the choice of whether to deepen the study of a certain promising enhancement, or to continue developing what I already had a grasp of. Amongst the topics that I certainly wanted to explore further there are:

- Dialog personalization based on user's known characteristics. First of all I could try to split the memory between dialog history and user's characteristics [18]. After that, it would be very interesting to implement a more sophisticated Memory Network extension such as the one presented by Luo *et al.* [19], called *Personalized MemN2N*, reported in Figure 4.24.
- Exploiting preexisting word embedding in addition to the ones learned by the model during training: it could be fundamental both to better capture the semantics of sentences and words, and most importantly to handle words that the model has never seen before.
- Generative approaches for Dialog Management System responses.

Finally, further small improvements could be attempted by experimenting with more refined tokenization methods or by trying to consider word ordering in sentence embedding, rather than coding them as bag of words.

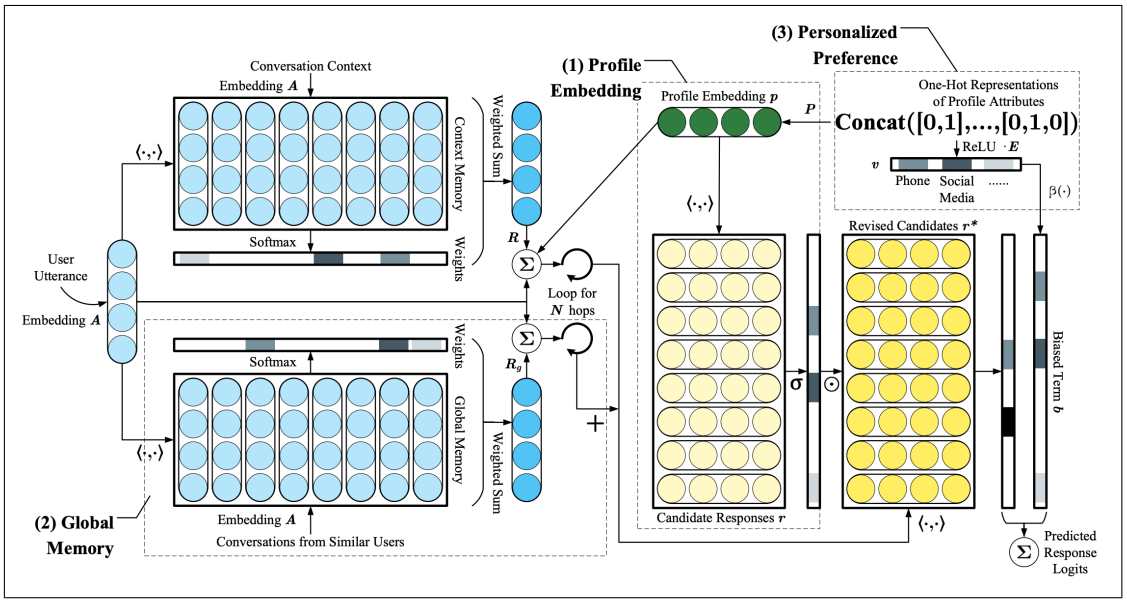

Figure 4.24: *Personalized MemN2N* architecture [19].

Chapter 5

Conclusions

During my time at IBM I studied the topic of End-to-End Goal-Oriented conversational agents, aiming at familiarizing myself with new solutions for Human-Computer Interaction. Market demand for virtual assistants is constantly growing, and the End-to-End solutions based on neural network models are attracting a lot of interest from the scientific community (especially after the recent successes in chit-chat settings) because, being trainable directly from dialog corpora, they allow to reuse the same solution on different application domains without the need for manual intervention by the designer.

After having thoroughly investigated the literature published in the last 5 years I chose a reference publication deemed particularly relevant [1], I studied it in depth and I implemented it exploiting the latest technologies available to me. I verified my implementation's proper functioning by comparing my results with those reported in the paper for the reference *Dialog bAbI dataset*; I therefore proceeded to generate a corpora of conversations similar to the Dialog bAbI but applied to the domain of risk awareness. The creation of the dataset was a useful exercise firstly to grasp a better understanding of some choices made in the original study, but also to comprehend the actual level of usability, the potential and the limitations of the developed model. It would have been interesting to use a set of real dialogues, but unfortunately I didn't have any. To obtain a complete PoC, I then developed a simple Web client that incorporated an embodied virtual agent capable of interacting with the user through voice, and was able to ask the pre-trained dialog model for proper answers to give to the user.

I appreciated and found stimulating the choice of my company supervisor to let me experiment with an emerging technology which prescinds from the application domain, rather than making

me invest time in developing a Goal-Oriented conversational agent the typical way through tedious, domain specific, manual modeling of the dialogue flow. It's been a valuable experiment to truly understand the basics one of the most promising frameworks for End-to-End Goal-Oriented dialog systems development, which could radically change the way personal assistants are produced in the next years with the advancement of research and technology.

This internship has helped me increasing confidence in my abilities and gave me basic knowledge of neural network development, an area of artificial intelligence which I'm very interested in and would love to continue working on. In conclusion, it's been an experience that I'd certainly do again, my time at IBM has been absolutely positive for three main reasons:

- the technologies used and the topics covered have been cutting edge and of strong personal interest;
- both colleagues and supervisors have always been helpful and extremely competent;
- the internship and thesis activities, despite rather short, have certainly enriched my personal and educational path.

Bibliography

[1] Antoine Bordes, Y-Lan Boureau, and Jason Weston. Learning end-to-end goal-oriented dialog, 2016.

[2] DigiEduHack. One day. Hackathons all around Europe and beyond. Be a part of it. `https://digieduhack.com/en/`, 2019. [Online; accessed 13-February-2020].

[3] Premiazione DigiEduHack. Progetto VirTuS - Virtual Tutoring and Simulation. `https://digieduhack.com/en/solutions/virtus-virtual-tutoring-and-simulation`, 2019. [Online; accessed 13-February-2020].

[4] Daniel Jurafsky and James H. Martin. *Dialogue Systems and Chatbots*, chapter 26, pages 1–35. Stanford University, 2019.

[5] WealthWizards. MyEva: your personal digital financial adviser. `https://myeva.com/`. [Online; accessed 10-February-2020].

[6] DoNotPay - The world's first Robot Lawyer. `https://donotpay.com/`. [Online; accessed 10-February-2020].

[7] Microsoft. XiaoIce. `https://www.msxiaobing.com/`. [Online; accessed 10-February-2020].

[8] James Lester, Karl Branting, and Bradford Mott. Conversational agents. *The Practical Handbook of Internet Computing*, pages 220–240, 2004.

[9] Christian Muise, Tathagata Chakraborti, Shubham Agarwal, Ondrej Bajgar, Arunima Chaudhary, Luis A Lastras-Montano, Josef Ondrej, Miroslav Vodolan, and Charlie Wiecha. Planning for goal-oriented dialogue systems. *arXiv preprint arXiv:1910.08137*, 2019.

[10] Pearl Brereton, Barbara A Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4):571–583, 2007.

[11] Jason D. Williams, Kavosh Asadi, and Geoffrey Zweig. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. *CoRR*, abs/1702.03274, 2017.

[12] Tiancheng Zhao and Maxine Eskénazi. Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. *CoRR*, abs/1606.02560, 2016.

[13] Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. End-to-end reinforcement learning of dialogue agents for information access. *CoRR*, abs/1609.00777, 2016.

[14] Bing Liu and Ian Lane. Adversarial learning of task-oriented neural dialog models. *arXiv preprint arXiv:1805.11762*, 2018.

[15] Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. End-to-end task-completion neural dialogue systems. *arXiv preprint arXiv:1703.01008*, 2017.

[16] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[17] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[18] Chaitanya K. Joshi, Fei Mi, and Boi Faltings. Personalization in goal-oriented dialog. *CoRR*, abs/1706.07503, 2017.

[19] Liangchen Luo, Wenhao Huang, Qi Zeng, Zaiqing Nie, and Xu Sun. Learning personalized end-to-end goal-oriented dialog. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6794–6801, 2019.

[20] Andrea Madotto, Chien-Sheng Wu, and Pascale Fung. Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. *CoRR*, abs/1804.08217, 2018.

[21] Bing Yu, Fuji Ren, and Yanwei Bao. Memory-to-sequence learning with lstm joint decoding for task-oriented dialogue systems. In *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 200–204. IEEE, 2019.

[22] Zheng Zhang, Minlie Huang, Zhongzhou Zhao, Feng Ji, Haiqing Chen, and Xiaoyan Zhu. Memory-augmented dialogue management for task-oriented dialogue systems. *ACM Transactions on Information Systems (TOIS)*, 37(3):1–30, 2019.

[23] Mihail Eric and Christopher D. Manning. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue, 2017.

[24] Wenqiang Lei, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He, and Dawei Yin. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1437–1447, 2018.

[25] Dieuwke Hupkes, Sanne Bouwmeester, and Raquel Fernández. Analysing the potential of seq-to-seq models for incremental interpretation in task-oriented dialogue. *CoRR*, abs/1808.09178, 2018.

[26] Libo Qin, Yijia Liu, Wanxiang Che, Haoyang Wen, Yangming Li, and Ting Liu. Entity-consistent end-to-end task-oriented dialogue system with kb retriever. *arXiv preprint arXiv:1909.06762*, 2019.

[27] Jason D Williams and Lars Liden. Demonstration of interactive teaching for end-to-end dialog control with hybrid code networks. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 82–85, 2017.

[28] Weiri Liang and Meng Yang. Hierarchical hybrid code networks for task-oriented dialogue. In *International Conference on Intelligent Computing*, pages 194–204. Springer, 2018.

[29] Julien Perez and Fei Liu. Gated end-to-end memory networks. *CoRR*, abs/1610.04211, 2016.

[30] Jason D Williams and Geoffrey Zweig. End-to-end lstm-based dialog control optimized with supervised and reinforcement learning. *arXiv preprint arXiv:1606.01269*, 2016.

[31] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve J. Young. A network-based end-to-end trainable task-oriented dialogue system. *CoRR*, abs/1604.04562, 2016.

[32] Bing Liu and Ian Lane. Iterative policy learning in end-to-end trainable task-oriented neural dialog models. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 482–489. IEEE, 2017.

[33] Bing Liu, Gokhan Tur, Dilek Hakkani-Tur, Pararth Shah, and Larry Heck. End-to-end optimization of task-oriented dialogue model with deep reinforcement learning. *arXiv preprint arXiv:1711.10712*, 2017.

[34] Da Tang, Xiujun Li, Jianfeng Gao, Chong Wang, Lihong Li, and Tony Jebara. Subgoal discovery for hierarchical dialogue policy learning. *arXiv preprint arXiv:1804.07855*, 2018.

[35] Zachary Lipton, Xiujun Li, Jianfeng Gao, Lihong Li, Faisal Ahmed, and Li Deng. Bbq-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[36] Bing Liu, Gokhan Tur, Dilek Hakkani-Tur, Pararth Shah, and Larry Heck. Dialogue learning with human teaching and feedback in end-to-end trainable task-oriented dialogue systems. *arXiv preprint arXiv:1804.06512*, 2018.

[37] Vladimir Ilievski, Claudiu Musat, Andreea Hossmann, and Michael Baeriswyl. Goal-oriented chatbot dialog management bootstrapping with transfer learning. *arXiv preprint arXiv:1802.00500*, 2018.

[38] Jiahuan Pei, Pengjie Ren, and Maarten de Rijke. A modular task-oriented dialogue system using a neural mixture-of-experts. *CoRR*, abs/1907.05346, 2019.

[39] Jamin Shin, Andrea Madotto, Minjoon Seo, and Pascale Fung. End-to-end question answering models for goal-oriented dialog learning. 2019.

[40] Weikang Wang, Jiajun Zhang, Qian Li, Mei-Yuh Hwang, Chengqing Zong, and Zhifei Li. Incremental learning from scratch for task-oriented dialogue systems. *CoRR*, abs/1906.04991, 2019.

[41] Stefan Constantin, Jan Niehues, and Alex Waibel. An end-to-end goal-oriented dialog system with a generative natural language response generation. In *9th International Workshop on Spoken Dialogue System Technology*, pages 209–219. Springer, 2019.

[42] Zhuoxuan Jiang, Ziming Huang, Dong Sheng Li, and Xian-Ling Mao. Dialogact2vec: Towards end-to-end dialogue agent by multi-task representation learning. *arXiv preprint arXiv:1911.04088*, 2019.

[43] Pawel Budzianowski and Ivan Vulic. Hello, it's GPT-2 - how can I help you? towards the use of pretrained language models for task-oriented dialogue systems. *CoRR*, abs/1907.05774, 2019.

[44] Corriere Comunicazioni. 38 mln di investimenti e 250 posti di lavoro, l'Emilia Romagna mette il turbo a Industria 4.0. `https://www.corrierecomunicazioni.it/industria-4-0/38-mln-di-investimenti-e-250-posti-di-lavoro-lemilia-romagna-mette-il-turbo-a-industria-4-0/`, 2018. [Online; accessed 7-January-2020].

[45] Bologna Today. Lavoro, intelligenza artificiale: IBM cerca laureati in materie tecnico-scientifiche. `http://www.bolognatoday.it/economia/offerte-di-lavoro/ibm-ingegneri-laureati-lavoro-assume.html`, 2018. [Online; accessed 7-January-2020].

[46] Corriere di Bologna. L'intelligenza artificiale sbarca in città: da IBM 5 milioni e 20 posti di lavoro. `https://corrieredibologna.corriere.it/bologna/economia/19_aprile_11/intelligenza-artificiale-sbarca-cittada-ibm-5-milioni-20-posti-lavoro-14f61552-5c32-11e9-b21f-a9bf73d2542a.shtml`, 2019. [Online; accessed 7-January-2020].

[47] ANSA. IBM Italia con UniBo, investe 5 milioni su ricerca. `http://www.ansa.it/emiliaromagna/notizie/2019/04/10/ibm-italia-con-unibo-5-mln-per-ricerca_6d0c03fd-7663-41d5-a267-546db73e2995.html`, 2019. [Online; accessed 7-January-2020].

[48] Humanitas. Research Hospital. `https://www.humanitas.it/`.

[49] Yun-Nung Chen, Asli Celikyilmaz, and Dilek Hakkani-Tur. Deep learning for dialogue systems. In *Proceedings of the 27th International Conference on Computational Linguistics: Tutorial Abstracts*, pages 25–31, 2018.

[50] Nikola Mrksic, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve J. Young. Neural belief tracker: Data-driven dialogue state tracking. *CoRR*, abs/1606.03777, 2016.

[51] Pei-Hao Su, Milica Gasic, Nikola Mrksic, Lina Rojas-Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. Continuously learning neural dialogue management. *arXiv preprint arXiv:1606.02689*, 2016.

[52] Andrea Madotto, Zhaojiang Lin, Chien-Sheng Wu, and Pascale Fung. Personalizing dialogue agents via meta-learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5454–5459, 2019.

[53] Facebook. bAbI tasks public datasets. `https://research.fb.com/downloads/babi/`, 2016. [Online; accessed 14-January-2020].