BRUNO EIDI NISHIMOTO

DEEP REINFORCEMENT LEARNING FOR MULTI-DOMAIN TASK-ORIENTED DIALOGUE SYSTEMS

São Paulo
2023

BRUNO EIDI NISHIMOTO

DEEP REINFORCEMENT LEARNING FOR MULTI-DOMAIN TASK-ORIENTED DIALOGUE SYSTEMS

REVISED VERSION

Manuscript submitted to the Escola Politécnica da Universidade de São Paulo to fulfill part of the requirements to obtain the degree of Master of Science in the Electrical Engineering Program.

São Paulo
2023

BRUNO EIDI NISHIMOTO

DEEP REINFORCEMENT LEARNING FOR MULTI-DOMAIN TASK-ORIENTED DIALOGUE SYSTEMS

REVISED VERSION

Manuscript submitted to the Escola Politécnica da Universidade de São Paulo to fulfill part of the requirements to obtain the degree of Master of Science in the Electrical Engineering Program.

Concentration Area: Computer Engineering

Advisor: Profa. Dra. Anna Helena Reali Costa

São Paulo
2023

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 31 de Janeiro de 2023

Assinatura do autor: Bruno Eidi Nishimoto

Assinatura do orientador:

Catalogação-na-publicação

Nishimoto, Bruno Eidi
Deep Reinforcement Learning for Multi-Domain Task-Oriented Dialogue Systems / B. E. Nishimoto -- versão corr. -- São Paulo, 2023.
65 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Aprendizado por Reforço 2.Chatbot I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

ACKNOWLEDGMENTS

First of all, I would like to thank my parents Wilson and Lúcia and my siblings Danilo and Fernanda who have always supported and trusted me in all my decisions throughout my life's journey.

A special thanks to my advisor Profa. Dra. Anna Helena Reali Costa for the advises and for allowing my evolution in the research area.

I thank Itaú Unibanco for the scholarship of the Itaú Scholarship Program (PBI) and assist me in participating in conferences, and Escola Politécnica for providing adequate infrastructure and environment for research.

I am grateful to the group of students who are part of Data Science Center (C2D) for the excellent interaction and the daily exchange of ideas.

Finally, I would like to thank everyone who somehow contributed to my evolution.

"Those who can imagine anything, can create the impossible."

-- Alan Turing

RESUMO

NISHIMOTO, B. E.**Deep reinforcement learning for multi-domain task-oriented dialogue systems**. 2022. Master Thesis. Escola Politécnica da Universidade de São Paulo, São Paulo, 2022.

O sistema de diálogo (DS) é uma ideia antiga que remonta a 1966, quando o primeiro sistema desse tipo foi criado. O DS pode ser classificado em três categorias: perguntas & respostas, orientado à objetivo e socialbot. Os sistemas de diálogo orientados à objetivo são um campo muito relevante devido à diversidade de aplicações possíveis que podem alcançar. Por exemplo, ele pode resolver tarefas como comprar um ingresso de cinema, reservar um restaurante e fornecer atendimento ao cliente. Eles têm recebido cada vez mais atenção nos últimos anos, e uma das razões para isso é o avanço no processamento de linguagem natural. Embora a literatura apresente diversos estudos com foco na SD, ainda há muitas questões a serem cumpridas. A maioria deles está relacionada à gestão do diálogo, componente central do DS. A aprendizagem por reforço (RL) é uma abordagem que alcançou grande sucesso recentemente. No entanto, as coisas se tornam mais complexas quando o DS é estendido para configurações de vários domínios, ou seja, quando o DS precisa concluir várias tarefas em diferentes domínios para o usuário. Alguns problemas como adaptação de políticas e transferência de aprendizado surgem nesse novo cenário. O objetivo desta pesquisa é aprimorar técnicas recentes de uso de RL na gestão do diálogo. Apresentamos um aprendizado eficiente equilibrando exploração e explotação, e potencializando o uso do conhecimento especializado para orientar o agente. Propomos um método para lidar com ruído e erro na entrada do gerenciamento de diálogo e também fornecemos uma comparação básica entre RL e aprendizado supervisionado em conjuntos de dados reais e de brinquedo. Por fim, apresentamos uma nova proposta para lidar com configurações multidomínio: o uso da técnica de dividir e conquistar e transferir aprendizado para diferentes domínios.

Palavras-Chave: *Chatbots*, Sistema de Diálogo, Gerenciamento de Diálogos, Aprendizado por Reforço, Múltiplos Domínios, Transferência de Aprendizado.

ABSTRACT

NISHIMOTO, B. E.**Deep reinforcement learning for multi-domain task-oriented dialogue systems**. 2022. Master Thesis. Escola Politécnica da Universidade de São Paulo, São Paulo, 2022.

Dialogue system (DS) is an old idea dating back to 1966, when the first such system was created. DS can be classified in three categories: question & answering, task-oriented and socialbot. Task-oriented dialogue systems are a very relevant field due to the diversity of possible applications it can achieve. For example, it can solve tasks like buying a movie ticket, booking a restaurant and providing customer service. They have received increasing attention in recent years, and one reason for this is the advancement in natural language processing. Although the literature presents several studies focusing on DS, there are still many issues to be accomplished. Most of them are related to dialogue management, the central component of DS. Reinforcement learning (RL) is one approach that has achieved great success recently. However, things become more complex when DS is extended to multi-domain settings, i.e. when DS needs to complete multiple tasks in different domains for the user. Some problems such as policy adaptation and transfer learning arise in this new scenario. The purpose of this research is to improve recent techniques using RL on the dialogue management. We present an efficient learning by balancing exploration and exploitation, and enhancing the usage of expert knowledge to guide the agent. We propose a method to handle noise and error in the input of the dialogue management and we also provide a basic comparison between RL and supervised learning in both toy and real datasets. Finally, we present a new proposal to deal with multi-domain settings: the use of the divide-and-conquer technique and transfer learning for different domains.

Keywords: Chatbots; Dialogue System; Dialogue Management; Reinforcement Learning; Multiple domains; Transfer Learning.

LIST OF FIGURES

LIST OF TABLES

LIST OF NOTATION SYMBOLS

a - an action

s - a state

$\mathcal{S}$ - set of all states

$\mathcal{A}$ - set of all available actions at every given state

$\mathcal{R}$ - set of all possible rewards

$\mathcal{T}$ - transition probability function

π - policy

π^* - optimal policy

q_π - state-action value function of policy π

t - time instant

S_t - state at instant t

A_t - action taken at instant t

r_t - reward at instant t

G_t - total accumulated reward until final instant T

LIST OF ACRONYMS

AdaGrad - Adaptive Gradients

CM - Confusion Module

CNN - Convolutional Neural Network

DCDA - Divide-and-Conquer Distributed Architecture

DM - Dialogue Management

DQN - Deep Q-Network

DRL - Deep Reinforcement Learning

DRPO - Deep Recurrent Partial Observable

DST - Dialogue State Tracking

GAE - Generalized Advantage Estimate

GDPL - General Data Protection Law

GRU - Gated Recurrent Unit

LSTM - Long Short Term Memory

MDP - Markov Decision Process

MSE - Mean Squared Error

NLG - Natural Language Generation

NLP - Natural Language Processing

NLU - Natural Language Understanding

POMDP - Partial Observable Markov Decision Process

PPO - Proximal Policy Optimization

SGD - Stochastic Gradient Descent

RL - Reinforcement Learning

RMSProp - Root Mean Squared Propagation

RNN - Recurrent Neural Network

S2M - Slot Sharing Mechanism

SL - Supervised Learning

t-SNE - t-Distributed Stochastic Neighbor Embedding

VMLE - Vanilla Maximum Likelihood Estimation

CONTENTS

1 INTRODUCTION

Dialogue systems or conversational agents, also known as chatbots, are computer programs that interact with humans employing natural language (SHAWAR; ATWELL, 2007). The idea of interacting with a machine using natural language was first introduced by Alan Turing, in 1950, with the Turing test (TURING, 2009). In this test, he presents the imitation game that defines a criterion to decide whether a machine is intelligent or not. Such a test, in a nutshell, comprises an interviewer and two participants, one of whom is a machine and the other a human. The interrogator does not know the identity of either and needs to ask a series of questions for both to do so. At the end of the interview, if the interrogator is still not able to figure out which of the participants is the human and which is the machine, then it is said that the machine passed the test and considers it being intelligent.

ELIZA was the first system to which the Turing test could be applied (WEIZENBAUM, 1966). It was created in 1966 by Joseph Weizenbaum from the Artificial Intelligence Laboratory at Massachusetts Institute of Technology. This system simulates the behaviour of a psychologist and responses are taken from a set of rule templates. Based on this and with the creation, in 1990, of the Loebner prize (POWERS, 1998), a competition based on the Turing test that evaluates and rewards the best chatbots, the number of studies concerned to this topic exploded, emphasizing the last years, in which growth was exponential.

Thenceforth, a substantial number of dialogue systems emerged and they can be arranged into three categories: question & answering, task-oriented, and socialbot (GAO; GALLEY; LI, 2018):

- **Questions & answering**: it usually consists of single-turn dialogues, in which the user asks a question and the system needs to correctly answer it. The answer must be objective and concise, and there is usually a knowledge base from which the answers are derived from.

- **Socialbot**: the system should have a pleasant and adequate conversation as long as possible with the user, with no specific purpose. The dialogues are open domain and can go through several subjects in the same dialogue. The major application of this type of dialogue system is entertainment.

- **Task-oriented**: the bot needs to complete a user task using the fewest number of interactions as possible. In this category, dialogues are generally closed domain, for example, restaurant reservations and purchase of movie tickets.

Recently, with the era of artificial intelligence, promising advances have been observed in both research and industry. Such advances resulted in the development of some popular dialogue-based systems such as Apple Siri[1] (2011), Amazon Alexa[2] (2014), Microsoft Cortana[3] (2014), Microsoft XiaoIce[4] (2014), and Google Assistant[5] (2016). Advancement in natural language processing (NLP) is one of the major factors that allowed the progress of conversational agents, making them able to better interpret the user's sentences and generate higher quality sentences, and thus, providing better conversations.

In an ideal world, a single agent should be able to operate in all three categories, such as shown in the example dialog of Figure 1. However, the great complexity of the problem caused researchers to focus on each category individually. In particular, task-oriented dialogue systems (CUAYÁHUITL; KEIZER; LEMON, 2015; LI; CHEN; LI, 2017; WEISZ et al., 2018; MO et al., 2018) have proven to be a topic of increasing relevance due to their diversity of applications, ranging from booking a restaurant to buying a movie ticket and providing customer service. Thus, they can help users to complete common daily tasks more conveniently and easily.

Moreover, such virtual assistants often provide services spanning over multiple domains at the same time, i.e., within a single dialogue the user can buy a plane ticket and book a hotel room in the destination city. Multi-domain dialogue systems (CUAYÁHUITL et al., 2016; CUAYÁHUITL et al., 2017; XU et al., 2020) are useful as they can complete multiple tasks for the user and especially because they can seize the information obtained in one domain to use in another domain. For example, if the system already knows that the user's flight destination city is New York, and he/she asks for a hotel, it should filter its search to hotels in New York unless specified by the user.

[1] https://www.apple.com/siri/
[2] https://www.alexa.com/
[3] https://www.microsoft.com/en-us/windows/cortana
[4] https://www.msxiaobing.com/
[5] https://assistant.google.com/

Turn	Dialogue
1	**usr:** Good morning!
	agt: Good morning, I hope you had a great weekend.
2	**usr:** not really.
	agr: You sound depressed. What can I do for you?
3	**usr:** I'd like to know where sales are lagging behind our forecast?
	agt: The worst region is [country], where sales are 15% below projections.
4	**usr:** Do you know why?
	usr: The forecast for [product] growth was overly optimistic.
5	**usr:** How can we turn this around?
	agt: Here are the 10 customers in [country] with the most growth potential, per our CRM model.
6	**usr:** Can you set up a meeting with the CTO of [company]?
	agt: Yes, I've set up a meeting with [person name] for next month when you are in [location].
7	**usr:** Thanks!

Figure 1: Example of an complete dialogue with the three categories. Turns 1 and 2 are socialbots (blue), turns 3 to 5 are question & answering (yellow), and turns 6 and 7 are task-oriented (red). Source: Gao, Galley and Li (2018)

Despite advances in NLP techniques, there are still some problems in the interactions with chatbots. Many of these problems are related to insufficient data as domain-specific dialogues are harder to obtain, to the lack of robustness for noisy inputs, and also to the complexity of tasks, especially for multi-domain systems (ZHANG et al., 2020). In multi-domain settings, there are, additionally, issues with domain adaptation (XU et al., 2020), that is, how to agilely adapt existing bots to new domains, and with sharing inter-domain knowledge, which consists in reusing some constraint information obtained during a specific subtask in another domain (SAHA et al., 2020).

This work focuses on using deep reinforcement learning (DRL) techniques on multi-domain task-oriented dialogue systems, more specifically on the dialogue management, the central component of the system that controls all the conversation flow.

1.1 Motivation

DRL technique has proved itself to be highly promising in recent years. Some successful use cases are: AlphaGO (SILVER et al., 2016), a game in which the machine won against the Go world champion Lee Sedol; OpenAI Five (OPENAI, 2018), a game in which 5 agents won 5×5 over an amateur team in Dota 2; AlphaStar (VINYALS et al., 2019) in which the machine could beat professional players in StarCraft II and the most recent AlphaFold (JUMPER et al., 2021), a highly accurate protein structure predictor.

Therefore, there is a trend in the adoption of DRL, which has already revealed remarkable success in the sphere of complex games. Thus, it is interesting to expand its applications to new domains such as dialogue systems. Although there are many works that already use DRL in dialogue systems (WANG et al., 2020; LI; CHEN; LI, 2017; GORDON-HALL et al., 2020), there are many open problems to be solved specially regarding transfer information in multi-domain dialogue systems. We believe that the use of DRL contributes to the high relevance of the proposal, besides being a very current research topic and of considerable importance in several application domains. For illustration, there is a strong concern in smart travel assistants: customer service chatbots that can not only book and schedule flights, but can integrate additional services via social media platforms; with open APIs, they can link supplementary services like *Airbnb* [6] and *Uber* [7]. Such example perfectly fits in the multi-domain topic as well. Chatbots can also handle smaller daily tasks, content delivery, personalized help with managing shopping lists, social life, etc. Anyhow, the applications are numerous, and this market is growing fast.

In addition, nowadays several simple tasks are done using a specific app for this purpose, for example, *Ifood* and *UberEats* are used to order food, *Cinemark* to make a purchase of a movie in the cinema, among others. A recurring problem with this is the need to download many apps, taking up a large portion of the memory. With conversational agents, this would not be necessary, since only a messaging application like *WhatsApp* or *Messenger*, for example, is enough to be able to accomplish such tasks.

Finally, it is already part of people's daily lives to use messaging applications for a long time. Therefore, they are already applied to interact by exchanging messages and so the learning curve for using chatbots would be practically null, increasing their accessibility, notably for people who have difficulties in handling more modern devices (the elderly, for

[6] ⟨https://en.airbnb.com/⟩
[7] ⟨https://www.uber.com/br/pt-br/⟩

example). Unlike apps, if you don't have a good user experience, its usage is not intuitive and painful to access. As a result of this work we expect to contribute in the advances of dialogue systems in a way to improve the quality of interactions with people and help them to accomplish specific tasks in a easier and faster way.

1.2 Objective

The major objective of this research is to improve state-of-the-art models for task-oriented dialogue systems operating on multi-domain scenarios. We will cover two relevant research gaps:

- **Complexity of Multi-Domain Systems:** it involves the use of the divide-and-conquer approach to create subsystems specialized to solve small problems, i.e., tasks in specific domains and that can solve the major problem when assembled all together. This enables the use of simpler and well known algorithms used in single domain dialogue systems.
- **Inter-domain knowledge sharing:** it relates to transfer information across domains so that the agent, in a single conversation, can reuse some overlapping information obtained while completing a specific subtask to achieve the success faster in another subtask.

Despite the main objective of this work, there are some other contributions related to improvements in the dialogue management for single domain systems that could help the construction of the system operating in the multi-domain setting. They include an efficient balance between exploration and exploitation, enhancement in the use of expert knowledge during training, handling eventual errors or noises in the dialogue's input, and a basic comparison between supervised and reinforcement learning approaches on dialogue management.

1.3 Organization of the Manuscript

This document is organized as follows: Chapter 2 presents the main conceptual aspects addressed in the research project, such as dialog systems, reinforcement and supervised learning. In Chapter 3 there is a brief presentation of related works in the area, indicating similarities and differences with our proposal. Then, Chapter 4 describes our proposal

and its implementation. Finally, Chapter 5 shows our results and some discussions and Chapter 6 determines our conclusions in this work and describes future paths in the area.

2 BACKGROUND

This chapter presents the main theoretical and conceptual aspects involved in the work. The first concept is an introduction to task-oriented dialogue systems, explaining their essential architecture, and going through each of the key components. Next, there is reinforcement learning (RL), starting from the intuition behind RL, its basic mathematical concepts, the types of algorithms, and subsequently a detailed view on deep Q-learning (DQN) and Proximal Policy Optmization (PPO) which are two classic algorithms in RL. Finally it shows the basic concepts of supervised learning and the transformers, one of its most recent architecture.

2.1 Task-Oriented Dialogue Systems

There are essentially two architectures of dialog systems: end-to-end and pipeline (or modular) (CHEN et al., 2017).

In the end-to-end approach (Figure 2), we model the system as a sequence-to-sequence problem, i.e., it receives the user's sentence in natural language as input and generates the system's response in natural language as an output. Thus, the chatbot is viewed as a single black-box component. The training of this method is carried out to learn a direct mapping between the natural language sentence on the user side to the natural language response of the system. Here, it is common to use supervised learning algorithms and in particular recurrent neural networks (WEN et al., 2017) with attention mechanisms (VLASOV; DRISSNER-SCHMID; NICHOL, 2018).

In the pipeline approach, there are three independent components: NLU (Natural Language Understanding), DM (Dialogue Management) and NLG (Natural Language Generation) (CHEN et al., 2017), as indicated in the Figure 3.

The DM comprises the Dialogue State Tracking (DST) and the policy. There are also some other combinations such as coupling NLU and DST (word-level DST) and coupling

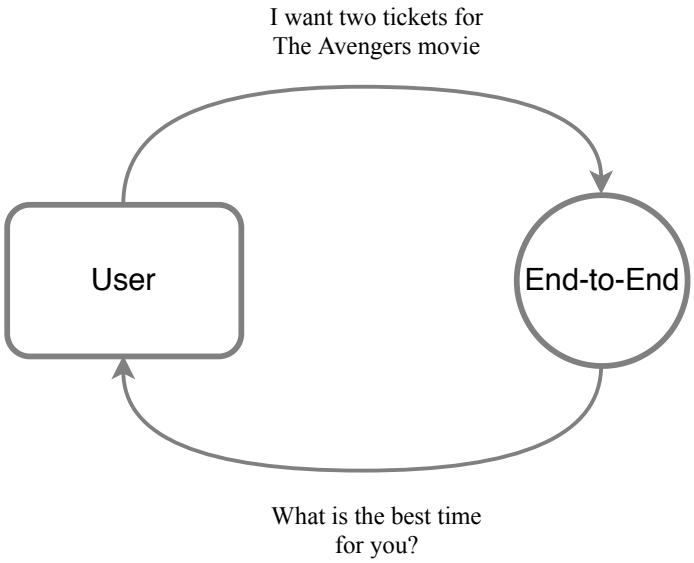


Figure 2: Simplified view of end-to-end architecture.

policy and NLG (word-level policy) (ZHANG et al., 2020).

The NLU is responsible for extracting important features from the user's utterance and transforming it into structured data so that the DM can process it. This structured data is known as a dialogue act and comprises intent, slots and domains (if multiple domains). All these info are predefined base on the specific domain(s). Intents represents the user's communication goal for that specific turn, e.g., `greeting`, `request` or `inform`. Slots are domain-specific keywords implemented in order to obtain a broader interpretation of the user's utterance. For instance, in the movie domain, some slots are `moviename`, `date`, `starttime` and `numberofpeople`. Both intents and slots are domain-specific and must be previously defined by domain experts. Formally, we represent the input as a word sequence $\vec{x} = \{w_1, w_2, \ldots, w_n\}$ and the output as a sequence of associated slots s_i for each word, the intent i and optionally the domain d (if in a multi-domain system), so that $\vec{y} = \{s_1, s_2, \ldots, s_n, i, d\}$. The intent and domain classification and the slot filling can be achieved either jointly or individually.

The DM is responsible for the entire flow of the conversation, that is, it chooses the best answer according to the current user action and the information previously got during the dialogue. There are two primary functions for the DM: dialogue state tracking (DST) and policy learning. The DST's function is to give a good estimation of the conversation state based on all interactions with the user so far. This is done by establishing and encoding all essential information of the dialogue, which includes slots already informed, the current turn number of the conversation, and the results of the database query. The policy module is the agent's brain. It encapsulates all the knowledge acquired during the

learning process. Its role is basically to map the state received by the DST to the fittest reply to the user. For example, it can request some specific information from the user, inform something important, or just wait for the user to provide more clues about its objective. A desirable feature for the DM is to deal with possible erroneous data from the NLU, that is, to be able to identify when the entry is possible wrong and work around this situation, either by asking the user to repeat or trying to discover the correct information.

Ultimately, NLG translates the action obtained from DM into a natural language utterance, providing a human-interpretable response. A good generator relies on several aspects such as adequacy, fluency, readability, and variation. Although the nature of these factors does not affect directly the performance of DM, they are desired in order to adapt the agent's language for each type of user, enhancing the system's usability.

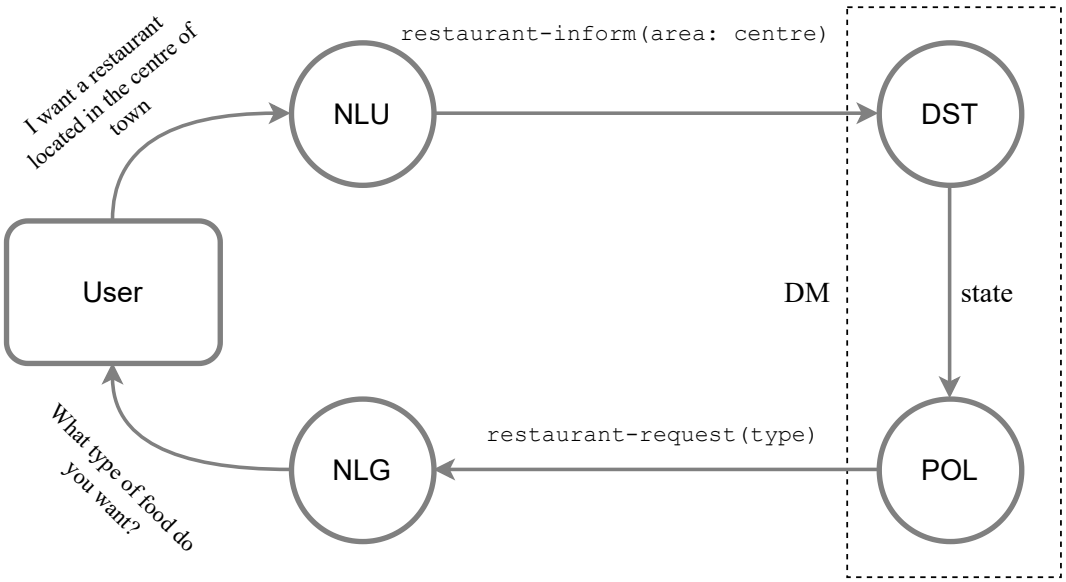


Figure 3: Simplified view of pipeline architecture.

2.2 Reinforcement Learning

Reinforcement learning is an area of machine learning in which the agent learns through interactions with the environment and rewards received for each taken action. It encompasses a collection of methods adopted to address sequential decision problems that are formalized by a Markov Decision Process (MDP) framework. An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$:

- $\mathcal{S} \rightarrow$ set of possible continuous or discrete states s of the environment. A state must be able to fully describe the situation of the agent in the environment;

- $\mathcal{A} \rightarrow$ set of possible continuous or discrete actions a that the agent can execute in a given state;
- $\mathcal{T} \rightarrow$ state transition probability function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, where $\mathcal{T}(s' \mid s, a) \doteq \mathbb{P}(S_t = s' \mid S_{t-1} = s, A_{t-1} = a)$ is the probability of the agent going to the state s' given that it was in the state s and executed action a. It describes the dynamics of the environment;
- $\mathcal{R} \rightarrow$ reward function $\mathcal{R} : \mathcal{S} \mapsto \mathbb{R}$ is the immediate reward the agent receives for being in state s.

The interaction between agent and environment is as follows: first the agent observes the state s_t of the environment and performs an action a_t. With a probability $\mathcal{T}(s_{t+1} \mid s_t,\ a_t)$ the environment goes to the state s_{t+1} and returns a reward $r_{t+1} = r(s_{t+1})$ for the agent — see Figure 4.

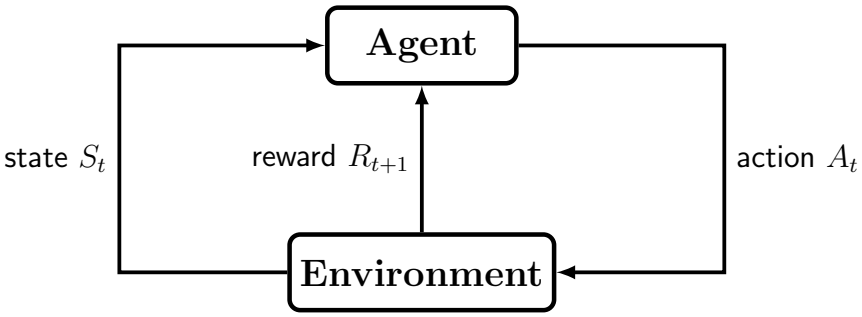


Figure 4: Interaction between agent and environment in an MDP framework.

The actions performed by the agent are selected according to the agent's policy π which can be either deterministic $\pi(s) : \mathcal{S} \mapsto \mathcal{A}$ or stochastic $\pi(s,\ a) : \mathcal{S} \times \mathcal{A} \mapsto [0,\ 1]$ based on a probability distribution over the actions.

In reinforcement learning, the agent's objective is to find an optimal policy π^*, which is the one that maximizes the expected return received in an episode. An episode is a sequence of actions taken by the agent through a sequence of states until reaching a terminal state. The return G_t is defined to be the total discounted cumulative reward the agent receives after time step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}. \tag{2.1}$$

where T is the final time step and γ is the discount factor whose intuition is to balance the importance of immediate and future rewards. That is, if $\gamma = 1$ then all rewards will

have the same weight. On the other hand, if $\gamma \approx 0$, then immediate rewards will have greater weight than future rewards. From a mathematical point of view, γ limits the return received in cases where the interaction has infinite horizon, since if $\gamma = 1$ and all rewards received are positive, then the return received will always be ∞.

Therefore, as the optimal policy π^* maximizes the expected return G_t, it can be formally defined as:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[G_t\right],$$

where $\mathbb{E}_{\pi}\left[G_t\right]$ is the expected return received by following policy π.

2.2.1 Exploration vs Exploitation

Before going into detail on the algorithms, one must understand two very important concepts: exploration and exploitation. Both are related to the agent's policy π and refer to how it chooses the action.

In exploration, the agent, without analyzing any metrics, chooses a random action. The idea is that, it needs to execute different actions and visit unknown states to have a better conception of the environment during the interactions. This enables the agent to possibly discover better actions.

On the other hand, exploitation explores the current knowledge of the agent, and performs the action that has given better results so far. This is important for the agent to refine its estimates and be able to receive high rewards and converge to an optimal policy.

Making a good balance between the use of exploration and exploitation is essential for efficient learning. At the beginning of interactions, as the agent has virtually no knowledge about the environment, it is preferable to use the exploration technique so it can discover new states and actions. After a long time interacting, the use of exploitation is advisable for the agent to improve the current estimates. The ϵ-greedy approach (SUTTON; BARTO, 1998) is a basic technique for this type of problem, which consists of the agent selecting a random action with probability ϵ, and choose the best action (greedy action) based on current estimates with probability $1 - \epsilon$

$$\pi(s) = \begin{cases} \text{random action from } \mathcal{A}, & \text{with probability } \epsilon, \\ \text{greedy action}, & \text{with probability } 1 - \epsilon. \end{cases}$$

2.2.2 Algorithms

RL algorithms can be splitted into model-free or model-based. In model-based algorithms, the agent learns the environment's transition $\mathcal{T}$ and reward $\mathcal{R}$ functions and then, with dynamic programming or other techniques, solves the problem using predictions from the learned models. However, it is very hard to learn a perfect model of the environment since the ground-truth of the environment is usually not available.

On the other side, in model-free algorithms, the agent does not care to learn the dynamics of the environment, i.e, it does not need to predict the next state and the reward to make its decision. It relies on experiences acquired through interactions with the environment to learn a policy. Because it does not need to learn a complete model of the environment, the later method ends up being more popular than the first one (ACHIAM, 2018).

Going deeper inside model-free group, we can find two groups: value-based and policy-based. Methods in the first one learn a function approximator $q(s, a;\ \theta)$ parameterized by θ for the optimal value function $q^*(s, a)$. The value function indicates the expected return the agent receives by executing action a at state s and then following the policy π,

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s,\ A_t = a \right]. \tag{2.2}$$

They are mostly off-policy algorithms, i.e., they can use experiences obtained from older policies and they usually need a memory replay to store their experiences and train the function approximator. Some examples of value-based algorithms are Q-learning, SARSA and DQN.

On the other side, policy-based algorithms directly optimizes the parameters θ of the policy $\pi_\theta(s|a)$ using gradient ascent on the objective function $J(\pi_\theta)$. These techniques are in majority on-policy, which means they only use data from the most recent version of the policy. A2C, A3C and PPO are some examples of such algorithms.

In this work, we use the Deep Q-learning (DQN) and Proximal Policy Optimization (PPO) algorithms. The first one is a value-based and off-policy algorithm and the later

a policy-based and on-policy algorithm. In the following, we explain in details both algorithms.

2.2.2.1 Deep Q-learning

DQN is based on the Q-learning algorithm that keeps the estimates of all value function $q(s, a)$ in a table and updates its values at each interaction:

$$q_{t+1}(S_t,\ A_t) \leftarrow q_t(S_t,\ A_t) + \alpha \left[\overbrace{\underbrace{R_{t+1} + \gamma \max_a q_t(S_{t+1},\ a)}_{target} - \underbrace{q_t(S_t,\ A_t)}_{estimate}}^{error} \right], \tag{2.3}$$

where α is the learning rate, which indicates the weight the agent gives to the new experience. Here, the expression $[R_{t+1} + \gamma \max_a q(S_{t+1},\ a) - q(S_t,\ A_t)]$ behaves like an error in the agent's estimate, where $q(S_t,\ A_t)$ is the current estimate, and $[R_{t+1} + \gamma \max_a q(S_{t+1},\ a)]$ would be the target.

However Q-learning and reinforcement learning algorithms are generally limited to simple problems. To solve complex problems, such as Go and chess with a huge number of states, it is impossible to store the value function of each state-action pair. A function that approximates the value function is then needed. For this, neural networks are used, introducing deep reinforcement learning.

Deep Q-learning, based on the classic Q-learning, is the first proposed deep reinforcement learning algorithm (MNIH et al., 2015). Instead of the table with all state-action pairs, it uses two neural networks, one as the training network to estimate the value function $q(s, a;\ \theta)$ and another as the target network $q(s, a;\ \theta^-)$. In the algorithm, the concept of experience replay is introduced, in which the agent stores the experiences $e_t = (S_t,\ A_t,\ R_t,\ S_{t+1})$ in memory $\mathcal{M}$. During learning, a random sample (batch) $(s,\ a,\ r,\ s') \sim U(\mathcal{M})$ is drawn from memory and the network weights are updated using the loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(\mathcal{M})} \left[(y - q(s, a;\ \theta_i))^2 \right], \tag{2.4}$$

where $y = r$ if s' is terminal state and $y = r + \gamma \max_{a'} q(s', a';\ \theta_i^-)$ otherwise, γ is the discount factor, θ_i^- are the weights relative to the target network that are used to calculate the error in the i-th iteration.

Algorithm 1 shows a pseudocode of *Deep Q-learning*. Mnih et al. (2015) evaluated the algorithm on several Atari games and showed to achieve a performance comparable to human players. First, we initialize the replay memory $\mathcal{M}$ and train $q(s, a;\ \theta)$ and target $q(s, a;\ \theta^-)$ networks. During each interaction, the agent observe state S_t and executes action A_t, receiving reward R_t and new state S_{t+1}. These experiences are stored in the replay memory $\mathcal{M}$. In order to update the parameters, we sample a batch of experiences from the experience replay memory $\mathcal{M}$. Then, it computes the target value $y \leftarrow r$ or $y \leftarrow r + \gamma \max_a q(s, a; \theta^-)$ depending if s is the final state or not, respectively. The parameters of the training network θ are updated by applying gradient descent on the loss function from Equation 2.4. The parameters θ^- from target network are updated every C (a predefined constant) episodes to have the same values as the parameters θ from training network. This process proceeds until there are no more episodes to run.

Algorithm 1 Deep Q-Learning with experience replay.

Initialize replay memory $\mathcal{M}$
Initialize training action-value function $q(s, a;\ \theta)$ with random weights θ
Initialize target action-value function $q(s, a;\ \theta^-)$ with weights $\theta^- = \theta$
repeat for each episode
    Initialize the environment and set up the first state S_1
    repeat for each step
        Choose action A_t in S_t following some exploration/exploitation strategy in $q(s_t, a; \theta)$
        Observe the reward r_t and new state s_{t+1}
        Store the experience (S_t, A_t, R_t, S_{t+1}) in replay memory $\mathcal{M}$
        Sample a random minibatch $\mathcal{B}$ of experiences (S_j, A_j, R_j, S_{j+1}) from $\mathcal{M}$
        repeat for each experience in $\mathcal{B}$
            if S_{j+1} is terminal state **then**
                set $y_j \leftarrow R_j$
            else
                set $y_j \leftarrow R_j + \gamma \max_a q(S_j, a;\ \theta^-)$
            end if
            Perform gradient descent with the loss function in Equation 2.4
        until no more elements in minibatch
        Every C episodes update $q(.; \theta^-) \leftarrow q(.; \theta)$
        Set $S_t \leftarrow S_{t+1}$
    until episode ends
until no more episodes

2.2.2.2 Proximal Policy Optimization

The PPO algorithm (SCHULMAN et al., 2017) is an online and policy based algorithm. Thus it estimates the policy with parameter θ and updates it by typically taking

multiple steps of Stochastic gradient descent (SGD) to maximize the objective L:

$$\theta_{t+1} \leftarrow \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{E} \left[L(s, a, \theta_t, \theta)\right], \tag{2.5}$$

where L is given by:

$$L(s, a, \theta_{old}, \theta) = \min\left(\rho(s, a, \theta)\hat{A}(s, a),\ clip(\rho(s, a, \theta), 1-\delta, 1+\delta)\hat{A}(s, a)\right), \tag{2.6}$$

where $\rho(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}(a|s)}}$ denotes the probability ratio, $\hat{A}$ is the estimated advantage function and δ is a hyperparameter that indicates how far can we go from old policy (although this symbol is usually ϵ in the literature, we used δ to distinguish from the ϵ-greedy policy).

For the advantage function estimation, there is the Generalized Advantage Estimation (GAE) (SCHULMAN et al., 2015), $\hat{A}(s_t, a_t) = \delta(s_t) + \gamma\lambda\hat{A}(s_{t+1}, a_{t+1})$, with $\delta(s_t) = r_t + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t)$, where γ is the discount factor, λ is a hyperparameter to adjust the bias-variance tradeoff and $\hat{V}(s)$ is the estimate of value function, i.e, the expected reward the agent receives being at state s.

Equation 2.7 shows the update rule for the value function $V(s)$ in the PPO algorithm, which is a regression on mean-squared error:

$$\phi_{k+1} \leftarrow \arg\min_{\phi} E\left[(V_\phi(s) - G)^2\right] \tag{2.7}$$

where G is the return from state s, that is, the sum of discounted reward received after state s.

Algorithm 2 describes the PPO algorithm. Schulman et al. (2017) tested the proposed PPO algorithm on several environments such as MuJoCo, Atari and other high-dimensional continuous control problems. First, it initializes the policy $\pi(s, a; \theta)$ and value function $V(s; \phi)$. Then, for each episode it collects a set of trajectories $\mathcal{D}_k = \{\tau_i\}$ and computes the return $\hat{R}$ and then the advantage estimate $\hat{A}$ for each time step. Finally it performs the update of the policy and value function using Equations 2.5 and 2.7, respectively.

2.2.3 Partially Observable Markov Decision Process

The MDP framework assumes the states are fully observable, that is, what the agent can observe is enough for the agent to have all the necessary information to know ex-

Algorithm 2 Proximal Policy Optimization.

1: Initialize policy $\pi(s, a; \theta)$ with random weights θ_0
2: Initialize value function $V(s; \phi)$ with random weights ϕ_0
3: **repeat** for k = 0, 1, 2, ...
4: Collect set of trajectories $\mathcal{D}_k = \tau_i$ by running policy $\pi(s, a; \theta_k)$ in the environment.
5: Compute rewards-to-go $\hat{R}_t$
6: Compute advantage estimates $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V(s; \phi_k)$
7: Update the policy parameters by following Equation 2.5
8: Fit value function applying regression on Equation 2.7
9: **until** no more episodes

actly how the environment is doing. However, there are cases where this does not happen, so a new formulation of the problem is needed. The partially observable Markov decision process (POMDP) (KAELBLING; LITTMAN; CASSANDRA, 1998) is a generalization of the MDP that emerged for this purpose. POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z})$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$ and $\mathcal{R}$ are the same as in MDP and:

- $\mathcal{O} \rightarrow$ set of observations. As the state is no longer observable by the agent, it acts according to the observations;
- $\mathcal{Z} \rightarrow$ is the conditional probability of observations $\mathcal{Z} : \mathcal{O} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,\ 1]$, where $Z(o \mid s,\ a) = \mathbb{P}(O_t = o \mid S_t = s,\ A_t = a)$.

As the state is not known by the agent, it must maintain a state of belief $\mathbf{b}$ which is a vector containing the probability that the agent is in each of the possible states $b(s_t) = \mathbb{P}(S_t = s_t)$. Thus, the policy defined by the agent is based on the state of belief rather than the state.

In the case of dialogue systems, this is an important factor to concern about, because the NLU component will not always capture the correct information from the user utterance. Hence, the dialogue management can not be sure about the dialogue action it receives, and can deal with these errors/noises by using the POMDP approach (GAŠIĆ et al., 2008; GAŠIĆ; YOUNG, 2014).

In our work, we studied how this issue affects the DM's performance when using MDP and POMDP by inducing random noisy on its input and comparing how the accuracy degrades as the level of induced noise increases.

2.3 Supervised Learning

Formally, in supervised learning, there is a dataset $\mathcal{D} = \{\mathbf{X}, y\}_N$ with N examples, where $\mathbf{X}$ represents a vector of values of some features of interest and y are the labels for each instance, that is, the correct answers for the corresponding data. y can be either discrete or continuous depending on whether the problem is classification or regression, respectively. The objective of the algorithm is to learn an approximation function $\hat{f}(\mathbf{X}) = y$ based on the dataset $\mathcal{D}$ that can generalize to new data not available in the dataset.

The quality of the approximation function is measured by the loss function or cost function $L(\hat{f}(X), y)$. It evaluates how far the model $\hat{f}$ is from the correct answer y, i.e., the higher its value, the worse the model is. Some standard loss functions in machine learning are: mean square error (MSE) or quadratic loss, hinge loss, and cross-entropy loss.

Given the loss function, the algorithm needs to minimize it to obtain a good approximation function to model the data. This optimization process is done with several different optimization algorithms, such as stochastic gradient descent (KIEFER; WOLFOWITZ, 1952) and its variants, root mean square propagation (RMSProp) (TIELEMAN; HINTON, 2012), adaptive gradient (AdaGrad) (DUCHI; HAZAN; SINGER, 2011), and adaptive momentum (AdaM) (KINGMA; BA, 2015). In general, these algorithms compute the directions that the loss function decreases and updates the parameters to follow these directions to reach a minimum and find the best model that suits the dataset.

In deep learning, a neural network represents the approximation function f. There are many types of neural network architecures, such as convolutional neural networks (CNN) (Lecun et al., 1998) and recurrent neural networks (RNN) (RUMELHART; HINTON; WILLIAMS, 1986). There is also the transformer architecture (VASWANI et al., 2017) which came out recently and replaced the traditional RNNs in many sequential tasks. A transformer architecture follows an encoder-decoder structure (CHO et al., 2014) which maps a sequence of input symbols $(x_1, x_2, \cdots x_n)$ to another sequence of output symbols $(y_1, y_2, \cdots y_m)$. The difference is that it does not rely on recurrence and convolutions to generate the output. Instead, it uses the attention mechanism (BAHDANAU; CHO; BENGIO, 2015). This attention mechanism allows the network to look at previous steps of the sequence and decide which of them are relevant for the current step. Figure 5 shows the architecture of a transformer model. On the left side there is the encoder composed of a stack o N identical layers, where each layer comprises a multi-head self-attention mechanism and a feed forward layer. On the right side, there is the decoder also

composed of N identical layers. Each layer is almost identical to the encoder layer with the addition of another multi-head attention layer, one of them computes the attention of encoder output and the other the attention of the decoder output of last time step. The attention function is computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.8}$$

where Q, K and V are query, key and values, respectively; Values V are vectors corresponding to each token[1] x_i of the input sentence $x = (x_1, \cdots, x_n)$; d_k represents the query and key vectors dimension. Intuitively, Q is the vector representation of the current input token x_i and K the vector representation of all other inputs in the sequence. So the function $softmax(.)$ computes the weights, that is, the relevance of each other input in the sequence related to the current input. Then, these weights are multiplied by their values V. Thus, the multi-head attention can be described as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \cdots, head_h)W^o, \tag{2.9}$$

where $head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and W denotes linear transformations. Therefore, the multi-head attention mechanism jointly consider information from different subspace of embedding, represented by the different linear transformations applied to the input vectors. As there are no recurrent units in this architectures, i.e., there is no order information, the positional encoding provides position information of the input. Finally the usual linear transformation and softmax function is applied to the decoder output to compute the model output probabilities.

In the context of dialogue systems, and more specifically of the DM, the input sequence is defined by the sequence of dialogue turns, where at each dialogue turn the input is featurized with the previous system action and the user actions (defined by the slots and entities) which represents the dialogue state. Therefore, the transformer attention mechanism can access different parts of dialogue history and learn their relevance from data. More details about how this is applied to DM is given in Section 4.4.

[1] the token represents each element of the sequence, for instance, in a text each word, space or special character is a token

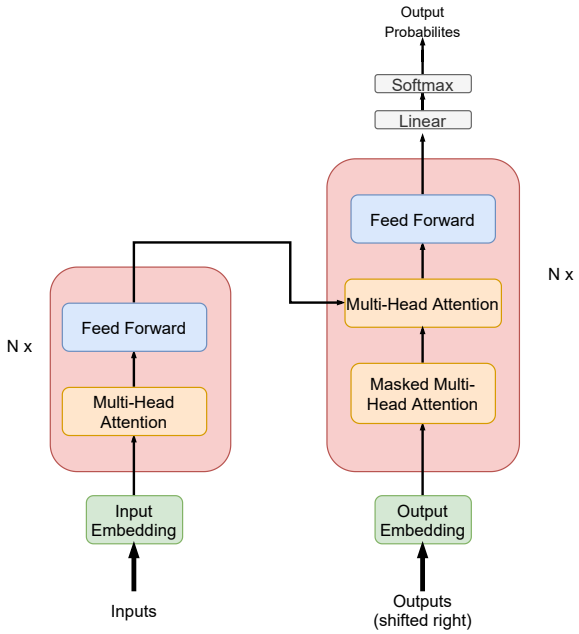


Figure 5: Transformer Architecture. Inspired in figure from (VASWANI et al., 2017)

.

3 RELATED WORK

We already remarked that there are three types of DSs. Here the focus is on the task-oriented ones and the two dominant methods researchers have adopted to solve this problem: RL and SL. We also present works on multi-domain scenarios for task-oriented systems.

Reinforcement Learning: to solve a DS using RL, we need to formalize it with the MDP framework to fit in a sequential decision problem (LEVIN; PIERACCINI; ECKERT, 1997; SINGH et al., 1999).

Prior works with value-based algorithms used linear methods to approximate Q-value. Li, Williams and Balakrishnan (2009), for example, employed a variation of policy iteration algorithm to optimize the linear model. However, linear methods require a set of predefined features and are too simple to model the dialogue. To deal with it, Gašić and Young (2014), Gašić et al. (2017) modeled the Q-value with a Gaussian Process, although they are computationally expensive. Recent works moved up to the neural networks. Li, Chen and Li (2017), Zhao and Eskenazi (2016) employed the classic deep Q-learning (DQN) algorithm to design a robust DM applying a multi-layer perceptron (MLP) to approximate the Q-function.

The second work (ZHAO; ESKENAZI, 2016) included an LSTM layer on top of the original network to support the manager deal with errors in the upstream module. Some other classical value-based algorithms have been assessed, including SARSA (MO et al., 2018) and deep dyna-Q (PENG et al., 2018b). Some other recent works also used the classical DQN algorithm to train the policy (GORDON-HALL; GORINSKI; COHEN, 2020; WANG et al., 2020), showing that despite simple, this algorithm can provide good results. (MO et al., 2018) and (WEISZ et al., 2018) tried out other RL algorithms to model the DM, such as SARSA and actor-critic, respectively.

Turning into policy gradient methods, Dhingra et al. (2017), Liu et al. (2018) selected the basic REINFORCE algorithm. The first one focused on getting better

queries from knowledge-base and the latter used human teaching and feedback in the loop to help the agent's learning. Rewards sparsity is a common issue related to on-policy and policy gradient algorithms. With this in mind, Takanobu, Zhu and Huang (2019) combined the PPO algorithm with a reward estimator based on inverse reinforcement learning. It uses samples collected from human dialog sessions and use these trajectories to optimize the reward estimator.

To avoid suboptimal behaviors and also to scale up training efficiency, various works carry out the use of human experts either before or during the training, since learning from scratch in RL is data and time consuming. The most simple yet effective method, used by off-policy algorithms, is to pre-fill the experience replay with some dialogues generated by an expert or a rule-based agent (LIPTON et al., 2018; LI; CHEN; LI, 2017). Further work pretrains the model using imitation learning with supervised learning before fine-tuning with RL. However, it is more complex and needs a lot of labeled data collected from experts. Su et al. (2016) simply trained the policy network to 'imitate' interactions from the corpus data. Peng et al. (2018a) employed adversarial learning where the discriminator needs to identify if an action is generated from the expert or the model. Moreover, the discriminator can be perceived as a reward function extracted from expert trajectories. Gordon-Hall, Gorinski and Cohen (2020) proposed de Deep-Q-learning from Demonstrations (DQfD), which uses expert demonstrations in a weakly supervised fashion.

Supervised Learning: With SL, the system is often trained in an end-to-end manner (we will refer to it as SL_{e2e}), that is, all three components (NLU, DM, and NLG) are considered being integrated into a single one. Then the SL_{e2e} algorithm learns to map a user utterance directly to a system response. A common approach for this problem is to use recurrent neural networks with the idea to retain information throughout the interactions. Vinyals and Le (2015) make use of a sequence-to-sequence framework, which is based on an RNN which reads the input token by token and the output is also predicted one token at a time. Serban et al. (2016), on the counterpart, mixed the encoder-decoder and hierarchical methods. The encoder maps the utterance to a hidden state, while the decoder performs the next utterance prediction. Bordes and Weston (2017) tested memory networks (WESTON; CHOPRA; BORDES, 2015) on goal-oriented chatbots, following their good performance on non goal-oriented chatbots (DODGE et al., 2016). The advantage of memory networks relies on the capability of storing previous dialogues and short-term context to reason about the response. In a more recent work, Lei et al. (2018)

used a LSTM with a CopyNet (GU et al., 2016) in a sequence-to-sequence model to learn a generation model that maps directly the user utterance into the system utterance. Wang et al. (2019) proposed an incremental dialogue system (IDS) which relies on bidirectional Gated Recurrent Unit (GRU) and an uncertainty estimation that shows how confident the model is on the response selection and select it if there is low uncertainty. It performs online learning together with human responses otherwise. Yang, Zhang and Erfani (2020) integrated graph knowledge into the end-to-end system and incorporate external knowledge bases to consistently improve the performance.

On the other hand, there are also some works that exploit SL in a modular manner to training the DM (which we refer to as SL_{pip}). Griol et al. (2008) used an MLP network to represent the dialogue management policy. Although Wen et al. (2017) developed its model in an end-to-end fashion, the architecture is modular, incorporating a feed-forward network as the dialogue policy which receives the state tracker, the user intents, and the database operation result. Bocklisch et al. (2017) adopted similar concepts to Hybrid Code Networks (HCN) (WILLIAMS; ASADI; ZWEIG, 2017) which is also a modular architecture trained with an end-to-end method, containing a RNN to compute the hidden state and a dense layer with softmax activation function to choose the next action. More recent works (VLASOV; DRISSNER-SCHMID; NICHOL, 2018; VLASOV; MOSIG; NICHOL, 2019) recommended the usage of embedding layer to produce an embedding representation of user input and system action with an attention mechanism over the RNN; and a transformer architecture to replace RNNs, respectively. Likewise, (HAM et al., 2020; HOSSEINI-ASL et al., 2020) used GPT-2, a pre-trained transformers model to build generative model in the pipeline architecture.

Multi-domain: most of the formerly reported works that are focused on single domains fail when extended to multi-domain settings due to the great growth in complexity. There have been some efforts in the past that focused on multi-domain task-oriented dialogue systems. Komatani et al. (2006) proposed a distributed architecture to integrate expert dialogue systems in different domains using a domain selector trained with a decision tree classifier. Further works employed traditional reinforcement learning to learn the domain selector (WANG et al., 2014). However, these systems require manual feature engineering for their building. Finally Cuayáhuitl et al. (2017) proposed to use deep reinforcement learning to allow training the system using raw data, without the manual feature engineering. Another common proce-

dure is the adoption of hierarchical reinforcement learning (CUAYÁHUITL et al., 2010). This technique normally consists of optimizing different policies in different hierarchies. Saha et al. (2020), for instance, trained in three different levels: domain level, intent level, and primitive action level. Some recent works give a great attention to centralized systems, i.e., a unique system capable of handling multiple-domains instead of having multiple agents, each one specialized for each domain. One reason for this is the increase in the power processing in modern computers. Some examples are the already mentioned works from Bordes and Weston (2017), Serban et al. (2016), Vlasov, Mosig and Nichol (2019). Despite these attempts to use supervised learning to learn a policy for the DM, RL is more used since it fits better in the sequential decision problem of a dialogue (DAI et al., 2020). Redundancy with respect to the overlapping slots between domains is another issue in multi-domain dialogue systems. Chen et al. (2019) address this problem by implementing the policy with a graph neural network where the nodes can communicate to each other to share information but they assume the adjacency matrix for this communication is known. Although recent work on multi-domain settings does not consider a distributed architecture, it is relevant for two reasons: it can reuse well-established algorithms for a single domain and, in the need to add a new domain to the system – which is common for real applications such as virtual assistants – it is unnecessary to retrain the entire system. Therefore, we adopted a distributed architecture using the divide-and-conquer approach. Furthermore, to the best of our knowledge, it is the first work that focuses on learning this slot-sharing mechanism.

Among all the works employing RL on the DM, most of them balance the exploration and exploitation with a very basic policy ϵ-greedy (with or without decay rate) (GAŠIĆ; YOUNG, 2014; LI; CHEN; LI, 2017; MO et al., 2018). Exploring other methods may enhance the learning efficiency of the agent. Another gap found is that just a few works are concerned with noise and error from the NLU component, most of them consider the information received by the DM is fully correct. Among all these works, only (LI; CHEN; LI, 2017) take advantage of a set of pre-defined rules, used in a warm-up phase before training. This method of expert knowledge is very cheap since it does not require any pre-training or a huge amount of labeled data and yet is very effective. In this work, we improve in the utilization of this knowledge, along with the acceleration of the agent's initial learning steps.

Despite the numerous proposals for designing a good DM, to the extent of our knowledge, few works in the literature compare the two paradigms in terms of their perfor-

mances. Vlasov, Mosig and Nichol (2019) made a comparison between modular SL (SL_{pip}) with end-to-end (SL_{e2e}) settings, while Takanobu et al. (2020) compared modular RL with end-to-end (SL_{e2e}). Both of them concluded that the modular approach achieves better results against end-to-end. However, it lacks in literature a comparison between modular SL (SL_{pip}) and modular RL. Furthermore, most works adopt data collected from small and controlled domains, where data is not a big issue for the training. In this work, we also cover these two gaps by comparing supervised and reinforcement learning applied to dialogue systems and also testing them in a real-world domain.

Table 1 compares our work with some other cited works in the dimension of multidomain systems, showing that we cover a topic which is not very explored but have high relevance.

Table 1: Related Works for Multidomain Dimension

Works	**Architecture**		**Redundant Information**	
	Centralized	Distributed	Manual Adj. Matrix	Learned Adj. Matrix
(BORDES; WESTON, 2017; SERBAN et al., 2016; VLASOV; MOSIG; NICHOL, 2019)	✓			
(CHEN et al., 2019)		✓	✓	
(KOMATANI et al., 2006; SAHA et al., 2020; CUAYÁHUITL et al., 2017)		✓		
Our Work		✓		✓

Table 2 shows some of the mentioned related works and their characteristics compared with what we did in our work: proposed a better exploration/exploitation technique, evaluated dialogue systems on real dataset, make a better use of expert knowledge without the need of pre-training, studied the effects of partially and fully observability on DM, compared the modular SL and RL approaches.

Table 2: Related Works in other dimensions

Works	**Exploration**		**Dataset**		**Expert Knowledge**			**Environment**		**Comparison**		
	ϵ-greedy	Softmax	Toy	Real	Simple Warm-up	Pre-training	Enhanced Warm-up	MDP	POMDP	RL vs end-to-end	SL vs end-to-end	RL vs SL
(LI; CHEN; LI, 2017; PENG et al., 2018b)	✓		✓		✓			✓				
(SU et al., 2016)	✓		✓		✓	✓		✓				
(GAŠIĆ; YOUNG, 2014)	✓		✓						✓			
(VLASOV; MOSIG; NICHOL, 2019)			✓								✓	
(TAKANOBU et al., 2020)			✓							✓		
Our work		✓	✓	✓			✓	✓	✓			✓

4 PROPOSAL

This chapter presents the proposal of this work to handle multi-domain dialogue systems. At first, we present the proposed architecture DCDA-S2M (Divide-and-Conquer Distributed Architecture with Slot Sharing Mechanism). It is composed of two parts, the Divide-and-Conquer Distributed Architecture (DCDA) that contains multiple agents trained individually for each domain and then ensemble them to compose the whole multi-domain system, and the Slot Sharing Mechanism (S2M), in which the system transfers information acquired during the dialogue across domains. Then, we present the node embedding approach used to implement the S2M and the reinforcement learning technique used to implement the DM and three improvements in relation to the article from Li, Chen and Li (2017) used as a guideline: a better balance between exploration and exploitation in dialogue systems and better management of the designer knowledge during training, and how to handle the noises and errors that come from NLU. Finally, we show some basic implementations of DM using supervised learning techniques to compare with the approach that uses reinforcement learning in DM.

4.1 Divide-and-Conquer Distributed Architecture

In the DCDA architecture, there are several agents, each one trained in a particular domain, as illustrated by the squares in Figure 7. Since multi-domain dialogue systems are much more complex compared to single domain systems due to the high increase in the state and action spaces, the use of DCDA would allow us to divide the complex problem into smaller ones and use simpler reinforcement learning algorithms to solve them.

Besides the multiple agents, the DCDA contains a controller that receives the input dialogue act with all the information (intent, slots and domain) from NLU. Based on the current domain of the dialogue, which is perceived directly in the dialogue act, the controller redirects the flow to the corresponding agent to collect the response. The state of the system is represented by a structured data, like a nested dictionary, where the

keys in the first level are the domains and in the second level the slots. Formally, it is represented as

$$\{d_i : \{s_{ij} : v_{ij}\}\} \quad \text{with } i = 1, 2, \ldots, D \text{ and } j = 1, 2, \ldots, |d_i|$$

where d_i represents each of the D domains of the system, s_{ij} represents the $|d_i|$ slots of the domain d_i and v_{ij} is a variable the contains the value of the slot s_{ij} which could be empty. Figure 6 shows an example of a state representation for a system with three domains: restaurant, hotel and attraction. So at each interaction, the controller receives the dialogue act from NLU and fills the corresponding domain and slot with their respective value.

restaurant: {
    pricerange: cheap,
    area: north,
    food:,
    people:
},
hotel: {
    area: north,
    pricerange: moderate,
    stars:,
    parking: no
},
attraction: {
    area: north,
    entrance_fee:
}

Figure 6: Example of the state representation in a system with $D = 3$ domains, where d_1 = restaurant, d_2 = hotel and d_3 = attraction. For the first domain $|d_1| = 4$ with s_{11} = pricerange, v_{11} = cheap, s_{12} = area, v_{12} = north, s_{13} = food, v_{13} = ‘ ’, s_{14} = people and v_{14} = ‘ ’. The same applies for the other domains.

The controller also contains a list $g = [g_1, g_2, \ldots g_K]$, where $g_i \in \{d_j\}_{j=1}^{D}$ is the i-th domain of the dialogue between the agent and the user. Therefore, at every turn, the controller checks if the domain of the dialogue act is different from the last element of

g, and if so, it detects a domain change and append the new domain to the list. In this way, the controller can keep track of all state features from all agents allowing it to know the past and current domains so that it can transfer knowledge across domain during a dialogue. For instance, in Figure 7 the current domain is `restaurant` as observed in the input dialogue act `restaurant-inform(area: centre)`. Thus the controller updates the dialogue states and check with the Slot Sharing Mechanism (explained in details in Section 4.2) if there is any information to be transferred from past domains. Finally it redirects the flow to the restaurant agent to give the response.

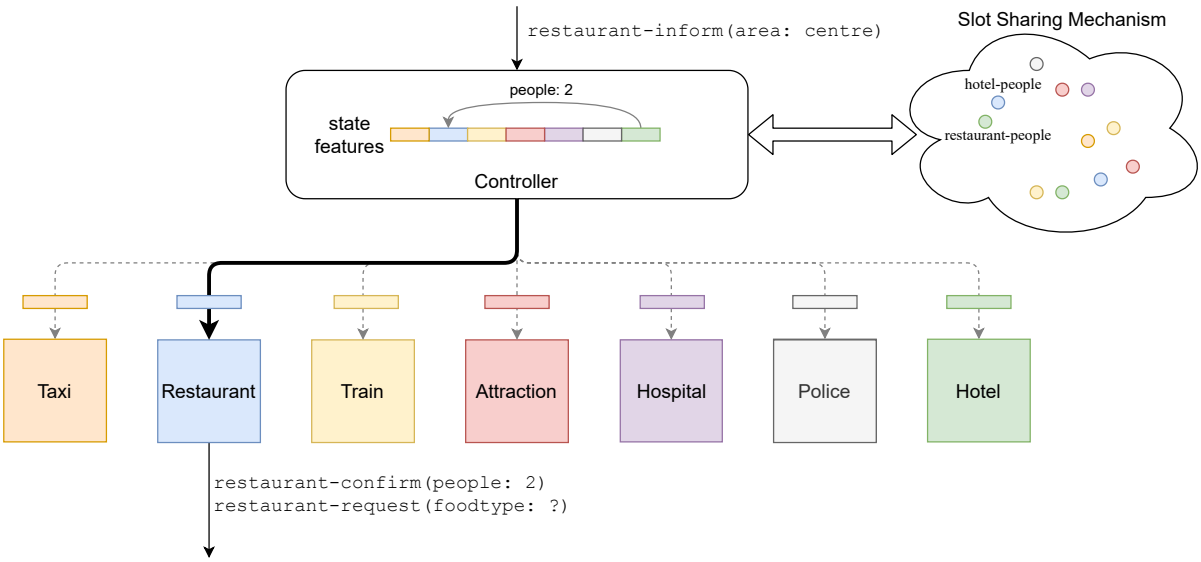


Figure 7: Illustrative figure of the proposed DCDA-S2M architecture.

4.2 Slot Sharing Mechanism

Some domains contain overlapping slots, i.e., slots that are likely to have the same value. For instance, if the user is looking for a restaurant and a hotel, it is likely that they are for the same day and in the same price range. There are also more complex relationships such as the time of reservation for a restaurant and the time when the taxi must arrive at its destination (which would be the restaurant). However, not all slots are shareable between two domains, so we need to know which slots whose content can be transferred from one domain to another.

In the following subsections we show how we learned these relationships. But for now, suppose we already know what such slots are. Therefore, during the conversation, when the controller notices a domain change, the slot-sharing mechanism first gets all informed slots in previous domains and then checks whether any of those slots can share

their values with any slots of the new domain. In this case, the controller transfers the information obtained during conversation to the new domain by copying the slots values. For example, suppose the system has already interacted with the user in the hotel domain and knows that the demand is for two people. When the conversation changes to the restaurant domain, the slot sharing mechanism will see that *hotel-people* slot can be shared with *restaurant-people* slot and the controller will transfer this information (two people) to the restaurant domain. The agent then asks for confirmation and acts considering this transferred slot. This can speed up the dialogue and improve the user experience during interactions by avoiding asking redundant information. In the worst case, if the transferred value is wrong, the user informs the correct value and continues the interaction normally.

4.2.1 Learning Shareable Slots

Our proposal to learn which slots can be shared, named Node Embedding for Slots Sharing Mechanism, uses the node embedding technique in which each node represents a *domain-slot* pair. The similarity of two nodes indicates whether they can share the same value in a conversation. It is defined by a simple scalar product, that is, given nodes $u = [u_1, u_2, \ldots, u_d]$ and $v = [v_1, v_2, \ldots, v_d]$, we have: $similarity(u, v) = \langle u, v \rangle = \sum_{i=1}^{d} u_i \cdot v_i$, where d is the embedding dimension. For instance, the nodes *restaurant-day* and *hotel-day* must be similar, i.e., have a high scalar product, while *restaurant-name* and *hotel-name* must have a low scalar product.

Before learning the node embedding, we need to build a similarity matrix $A \in \mathbb{R}^{n \times n}$, a $n \times n$ matrix where n is the number of nodes, i.e, number of *domain-slot* pairs and each cell A_{uv} represents the similarity between nodes u and v normalized to range between 0 and 1, that is, $similarity(u, v) \in [0, 1]$. This is done using the dialogues from dataset $\mathcal{D}$ as shown in Algorithm 3. At the end of each conversation, we observe the final state (the state of the dialogue in the last interaction) and check if each node pair have the same value (a node represents a *domain-slot* pair). If they do, the weight between these two nodes is increased by one. In the end, all weights are normalized to the number of times each pair of nodes appeared in the dialogues.

However, keeping a matrix with $\mathcal{O}(n^2)$ of space complexity does not scale with the number of domains and slots. For this reason, we trained a node embedding representation for each *domain-slot* pair. The learning of node embedding uses the similarity matrix $A \in \mathbb{R}^{n \times n}$ and follows Algorithm 4 proposed by Ahmed et al. (2013). In each step, for

Algorithm 3 Similarity Matrix

Require: Dataset $\mathcal{D}$

Initialize similarity matrix $A \in \mathbb{R}^{n \times n}$ with zeros
Initialize pair-counter matrix $C \in \mathbb{R}^{n \times n}$ with zeros
for all dialogue $d \in \mathcal{D}$ **do**
    Set $goal \leftarrow$ final state of dialogue d
    for all $(u, v) \in goal$, where u and v form a *domain-slot* pair **do**
        if $value(u) = value(v)$ **then**
            $A_{uv} \leftarrow A_{uv} + 1$
        end if
        $C_{uv} \leftarrow C_{uv} + 1$
    end for
end for
$A \leftarrow A/C$ ▷ normalize similarity matrix
return A

each node pair $(u, v) \in E$, where E is the set containing all node pairs, it performs an update to minimize the following error

$$L(A, Z, \lambda) = \frac{1}{2} \sum_{(u,v) \in E} (A_{uv} - \langle Z_u, Z_v \rangle)^2 + \frac{\lambda}{2} \sum_u ||Z_u||^2 \tag{4.1}$$

The gradient of this loss function with respect to Z_u is given by:

$$\frac{\partial L}{\partial Z_u} = - \sum_{(v) \in N(u} (A_{uv} - \langle Z_u, Z_v \rangle) Z_v + \lambda Z_u. \tag{4.2}$$

where Z represents the embedding space and Z_u is the vector for node u. Intuitively, $\langle Z_u, Z_v \rangle$ is the similarity between Z_u and Z_v and then this algorithm wants to get $\langle Z_u, Z_v \rangle$ as close as to A_{uv} which represents the similarity taken from the similarity matrix with λZ_u as a regularizer factor. The t in Algorithm 4 can be thought as a learning rate for each update.

Therefore, the S2M contains a node embedding Z_u of each *domain-slot* pair u (node) and can infer the similarity of all pair of nodes (u, v) by computing the dot product of their corresponding node embedding $\langle Z_u, Z_v \rangle$. Furthermore the S2M is independent of the agent trained, so it can be plugged with any agent we want.

4.2.2 DCDA-S2M Dialogue Management

Gathering the DCDA, the S2M and all the agents we get the DCDA-S2M archicture for the DM (Figure 7). The flow inside this architecture is as follow: at each interaction, the controller gets the dialogue act from NLU and checks if its domain d_u is different from

Algorithm 4 Node Embedding

Require: Matrix $A \in \mathbb{R}^{n \times n}$, embedding dimension d, regularization factor λ, set of all node pairs E

Initialize $Z \in \mathbb{R}^{n \times d}$ at random
$t \leftarrow 1$
repeat
  for all $(u, v) \in E$ **do**
    $\eta \leftarrow \frac{1}{\sqrt{t}}$
    $t \leftarrow t + 1$
    $Z_u \leftarrow Z_u + \eta \left[(A_{uv} - \langle Z_u, Z_v \rangle) Z_v \right] + \lambda Z_u$
  end for
until no more epochs
return Z

the current domain (which is the last element of the list of domains g). If so, it detects a domain shift and appends d_u to g. After that, the controller updates its state features by matching the keys domain and slots to fill with the value received in the dialogue act from NLU. If there is no domain change, the controller redirects the flow directly to the corresponding agent of the current domain. Otherwise, the controller asks the S2M for all slots of past domains that can share values with slots from the current domain. Then, for each slot, the S2M computes the similarities and returns a list of all *domain-slot* pairs from old domains that can share value with it. Finally the controller fills the slots of the new domain with values from shareable slots from past domains and then redirect the flow to the corresponding agent. Inside each agent, the state features are encoded as detailed in the next section and the agent's policy returns the response to the user.

4.3 Reinforcement Learning in Dialogue Systems

This section presents the implementation of the DM used in this work, which is based on Li, Chen and Li (2017), where the authors used the DQN algorithm (MNIH et al., 2015) with an ϵ-greedy policy to balance exploration and exploitation during training.

Figure 8 summarizes the architecture used in the implementation and Algorithm 1 (see Section 2.2.2.1) shows the steps followed to train the DM. The dashed and green components in figure are only used during the training phase.

The architecture of the DQN algorithm contains a user simulator (right part of the figure), which represents the environment and interacts with the agent following the MDP framework. The user action u_a passes through the DST module and it returns the dialogue state s. During the interactions, the agent acts either by choosing the action based either

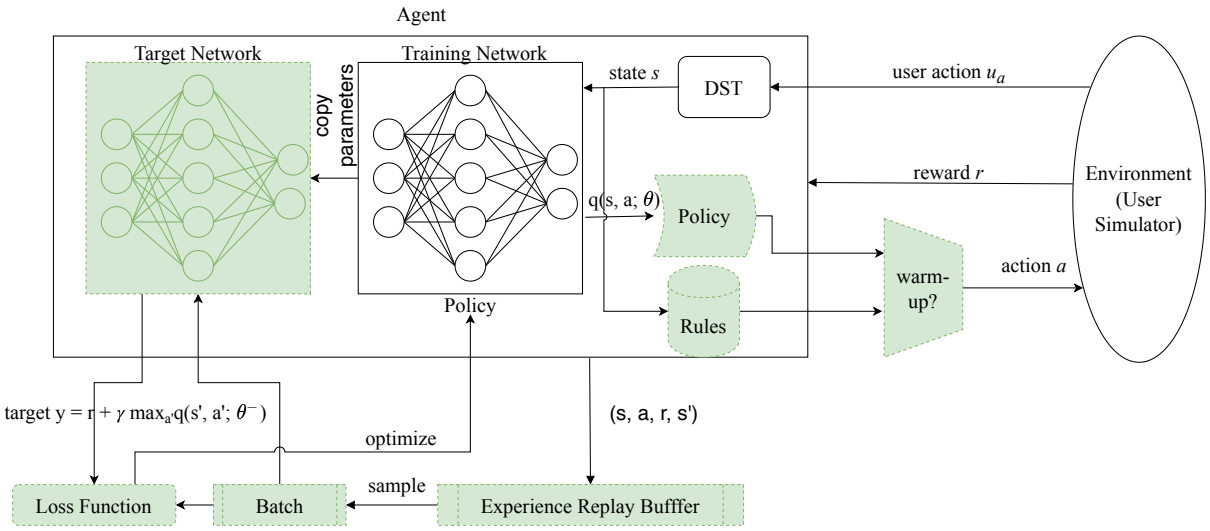


Figure 8: Architecture of the DQN algorithm. The dashed and green components are only used during the training phase.

on a rule based agent pre-defined by the system designer — if it is in the warm-up phase — or by adopting a given policy (ϵ-greedy, softmax, etc.) concerning the training action-value function $q(s, a;\ \theta)$ — if it is in the learning phase. The Q-value is computed by a forward step of the training network. After receiving the agent action, the user simulator gives a reward r and the next user action u'_a, transitioning the dialogue to the next state s'. During both the warm-up and training phase, the agent stores the interactions (s, a, r, s') in the replay buffer $\mathcal{M}$. These interactions are then used to update the training network parameters θ during the learning phase. For this, we sample a batch of experiences (s, a, r, s') from the replay memory $\mathcal{M}$ and compute the target value,

$$y \leftarrow \begin{cases} r + \gamma \max_{a'} q(s', a';\ \theta^-), & \text{if s is not a terminal state,} \\ r, & \text{otherwise.} \end{cases} \tag{4.3}$$

These targets are inspired on the Bellman's equation (BELLMAN, 1957) and they use the target action-value function $q(s, a;\ \theta)$ (from target network) to calculate them. The parameters θ of the training network are updated by applying gradient descent on the loss function in Equation 2.4.

The parameters θ^- from target network are updated every fixed number C of episodes to have the same values as the parameters θ from training network. That is, we copy the parameters values of training network into the target network every C episodes and remain them fixed between individual updates.

As seen in Section 2.2, in reinforcement learning, the agent learns by interacting with

the environment — in the case of chatbots, by talking to the users. However, a large number of interactions are required until the agent reaches an acceptable policy, making the training with human experts unfeasible. In order to avoid this issue, it requires a user simulator to replace humans to interact with the agent.

The user simulator follows an agenda-based approach (SCHATZMANN; YOUNG, 2009). At the beginning of the episode, it randomly sample a rule for the user from a rules list and then starts a conversation with the dialogue system using a rule-based policy. These rules are domain-specific, so for each domain there is one user simulator with different rules. However, in general, the user simulator contains a agenda with a stack-like structure that contains all slots needed to inform and request to complete the task. At each turn it pops the action based on this agenda until the task is completed.

Due to the necessity of encompassing all important information presented in the dialogue, the state representation s (output of the DST component) is composed of the dialog turn, the current user action, the last agent action, and the already-filled slots, i.e., all slots already reported or confirmed by the user. Both the user and the agent actions are formed by an intent and a collection of slots. The intents and slots are encoded with one-hot encoding. Figure 9 shows an example of intent encoding.

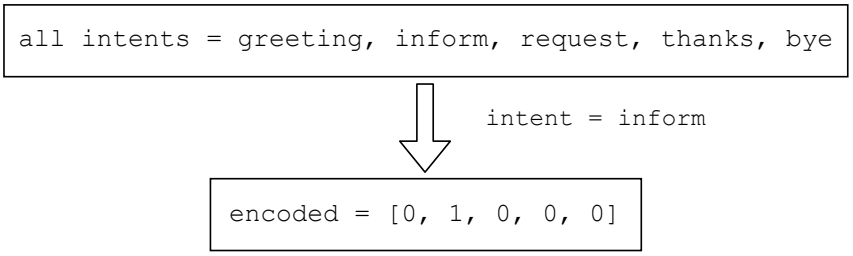


Figure 9: Example of how intents are encoded.

Finally, all the information are encoded and stacked together in a single array, forming the state representation.

The reward function is defined as follow:

$$R(s) = \begin{cases} R_{success}, & \text{if success,} \\ -R_{fail}, & \text{if fail or max round } M_{rounds} \text{ is reached,} \\ -1, & \text{otherwise.} \end{cases} \tag{4.4}$$

for each turn, i.e., for each action it takes, the agent receives a small penalty of -1. This penalty encourages the agent to achieve the goal as fast as possible. At the end of the

dialogue, if the goal is successfully achieved, the agent receives a larger reward of $R_{success}$, whereas if the goal is not successfully achieved, or the dialogue reaches the maximum number of rounds M_{rounds}, the agent receives a larger penalty of $-R_{fail}$.

The warm-up process is conducted in order to improve the agent learning with the help of the knowledge of experts represented in the form of pre-defined rules. It occurs in a stage before the training itself. During this phase, the agent acts by following a collection of expert-defined rules. It serves to fill the experience replay memory with these interactions that will help the agent in the latter policy training turns. This phase is important because in dialogue systems rewards are generally sparse. So learning from tabula rasa may be very difficult to the agent to find an optimal policy. Given that, the warm-up phase helps to guide the agent's initial learning with the knowledge of experts.

The memory replay buffer $\mathcal{M}$ is used to store the agent's experience acquired through interactions and to train the Q-network with these experiences in an off-policy manner. It has limited size $|\mathcal{M}|$, and the older experiences are replaced when it is full, similar to the behaviour of a circular list.

When testing, DQN does not use the dashed components from Figure 8. So the user actions u_a passes through the DST component and based on the state s the agent acts with a greedy policy, that is, it chooses the action with the highest Q-value.

4.3.1 Softmax

As discussed in the section 2.2.1, the way the agent balances between exploration and exploitation is very important for the agent to find the optimal global policy.

Li, Chen and Li (2017), the baseline article, uses an ϵ-greedy policy,

$$\pi(s) = \begin{cases} \text{random action from } A(s), & \text{if } \xi < \epsilon, \\ \arg\max_{a \in A(s)} Q(s,a), & \text{otherwise,} \end{cases}$$

where $\epsilon \in [0,1]$ is a hyperparameter indicating how much the agent will explore the environment and $\xi \in [0,1]$ is a random real number between 0 and 1 sampled at every step. This is a very basic policy used in most of the articles using DQN algorithm on dialogue systems in which basically the agent takes a random action (explore) with probability ϵ or takes a greedy action (exploit) with probability $1 - \epsilon$. The major issue related to this policy is that when exploring, the agent uniformly samples an action, giving

equal importance to all actions. This way, the agent can choose actions that it already knows for being 'bad' actions with the same probability as other actions. It can delay the learning of the agent as it is guided only by the rewards received, and 'bad' actions give low rewards.

Therefore, we proposed to use a softmax policy, which is a non-deterministic policy based on the distribution of *Boltzman* to classify each state-action pair (NISHIMOTO; REALI COSTA, 2019):

$$\pi(a \mid s) = Pr\{a_t = a \mid s_t = s\} = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_b e^{\frac{Q(s,b)}{\tau}}},$$

where parameter τ is called *temperature* and it indicates the degree of exploration of the agent. The higher its value the more exploration the agent makes, because softmax makes actions to be approximately equiprobable (TOKIC, 2010). The value of τ linearly decreases from $\tau_{\text{init}} = 2.0$ to $\tau_{\text{stop}} = 0.5$,

$$\tau = \max\left(\tau_{\text{stop}},\ \tau_{\text{init}} - \left(\frac{\tau_{\text{init}} - \tau_{\text{stop}}}{\tau_{\text{decay\_rate}}}\right) \cdot \text{epoch}\right),$$

with $\tau_{\text{decay\_rate}} = 40$ indicating at what epoch it will achieve τ_{stop} and an epoch is a set of 100 dialogues carried out by the agent.

4.3.2 Memory Flush

The strategy adopted in (LI; CHEN; LI, 2017) and (NISHIMOTO; REALI COSTA, 2019) resides in the flush of the experience replay when the agent achieves a certain success rate (30%) and then, during the training process, flush again every time the behavior network outperforms its previous policy. In other words, for every epoch, if the average success rate is greater than the current best success rate, then it flushes the memory. Li, Chen and Li (2017) argue that using data only from the best policy to train the network, that is, using the best experiences to train the network can lead to a better learning process. The major issue in this strategy, is that an early flush of the memory would loose all the warm-up interactions, that is, all the experiences with some expert knowledge embedded in it. Thus, there will be no more human guidance after the first flush.

In this work we propose to keep experience replay instead of flushing it. The intuition behind it is that the experience replay is filled with some rule-based actions which may

guide the agent's learning primarily during the early epochs of the training phase due to the pre-defined rules inherited from the warm-up process (NISHIMOTO et al., 2022). By not flushing the memory, we keep these expert defined rules as long as possible and help the agent to make a better use of them, thus learning a good policy faster than otherwise. Flushing the memory when the agent reaches a certain success rate threshold can lead the algorithm to not use properly the knowledge summed up in these rules.

4.3.3 Deep Recurrent Partially Observable Dialog Management

Section 2.1 states that the DM must be robust enough to handle with errors coming from the NLU. Thus, we also proposed to model the problem using POMDP instead of the classic MDP framework. In POMDP, the user action, as it has some degree of uncertainty, is treated as part of the agent's observation. However, uncertainties in the user's action also cause uncertainties in other components that comprise the state of the dialogue such as slots already informed, making the state not fully observable.

In this formulation, DM receives as input a list of size N that contains N possible actions of the user (instead of just the action provided by the NLU). Given a noisy entry $\tilde{a}_u$ from NLU that indicates the user action, a confusion model (CM) generates a list of N possible actions for $\tilde{a}_u$ so that DM can keep track of various dialogue paths during the conversation. Each action in this list may have a different set of informed slots. Hence we also need to maintain an N sized list of each one of these `inform` slots. Thus, for each action in the list, DST updates the filled `inform` slots and combine them with the last agent action and current turn. Finally, the final representation of the observation includes the N-list of user actions a_u^N, informed slots s_u^N, and the number of matches in database db^N, i.e., the number of entries in the database that match the current constraints (if applicable), as well as the last action a_s taken by the DM agent and the current turn t:

$$o_t = (a_u^N, s_u^N, db^N, a_s, t),$$

where the superscript N indicates a list o N values.

To deal with POMDP a recurring network is used in the *deep recurrent Q-learning* (HAUSKNECHT; STONE, 2015) algorithm. The intuition is that, as the dialogues have a sequential characteristic, the LSTM layer is able to capture this temporal information throughout the conversation. Therefore it will aggregate information over the dialog turns and may recognize possible errors, returning an approximation of the latent state space,

which is the actual belief state b_t. Finally the last layer is a dense layer that learns the policy approximation.

Figure 10 shows an abstract view of the proposal. CM is a confusion model that generates the N-best list based on the user action. It is formally defined as:

$$C(\tilde{a}_u) = [\tilde{a}_{u1}, \tilde{a}_{u2}, \cdots, \tilde{a}_{uN}],$$

where $\tilde{a}_{ui}$ is the i-th generated action and $\tilde{a}_{u1}$ is always the same as $\tilde{a}_u$.

There are basically four types of errors that can occur in the user action:

- **Intent error (I):** error on the classification of the intent;
- **Value error (S1):** error on the value of the slot;
- **Slot error (S2):** error on the slot name and slot value;
- **Miss slot (S3):** NLU skips a slot that should be classified.

Table 3 shows an example of each type of error. For each line, the part in *italic* is where the error occurred, and in the last line, the slot-value pair for `starttime` is missing. The error of `intent` is considered independent of others, i.e., it can occur regardless of the occurrence of the other three errors.

Table 3: Example of possible errors

Action
inform(moviename='Joker';starttime='7:30pm')
Intent error
request(moviename='Joker';starttime='7:30pm')
Value error
inform(moviename=*'The Avengers'*;starttime='7:30pm')
Slot error
inform(moviename='Joker';*numberofpeople='3'*)
Miss slot
inform(moviename='Joker')

When DM receives the input, CM creates $N-1$ noisy actions. Each action is generated by applying one of the slots errors (S1, S2 or S3) based on a probability distribution. Additionally it also generates an **I** error with some probability. The final list of actions N is composed of the received one and $N-1$ generated by the application of some noise.

This list can be seen as a N-best list that comes from NLU, hence there is high probability that the correct action is in it and we might assume that.

Our system was called DRPO-DM (*Deep Recurrent Partial Observable*) (NISHIMOTO; REALI COSTA, 2020).

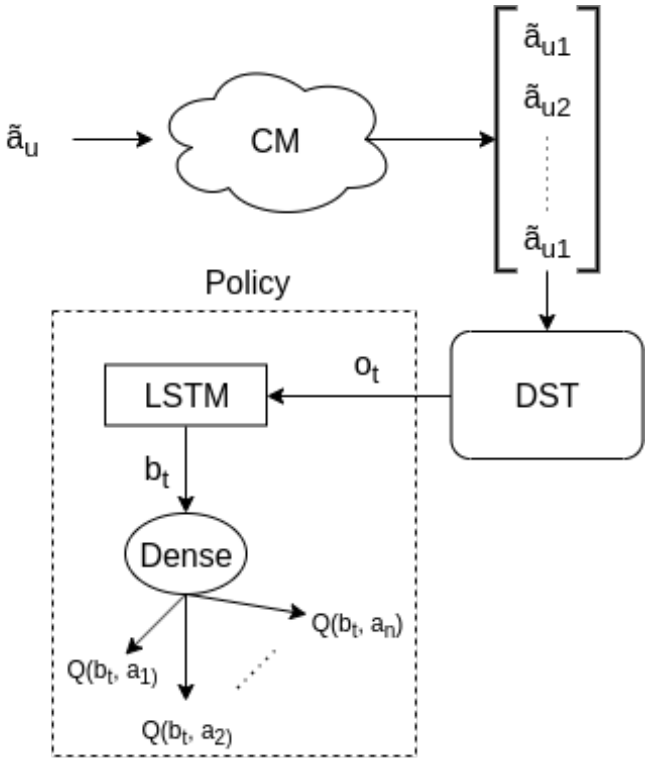


Figure 10: Sketched view of the DRPO-DM system. Source:*Deep Recurrent Q-Learning for Partially Observable Dialog System* (NISHIMOTO; REALI COSTA, 2020).

4.4 Supervised Learning in Dialogue Systems

Convolutional Neural Network (CNN), Gated Recurrent Unit (GRU), and Long Short Term Memory (LSTM) are common neural networks used in supervised learning (SL) and even used in some works focused on dialogue systems (GRIOL et al., 2008; WILLIAMS; ASADI; ZWEIG, 2017). However, as mentioned before, transformer architecture has become quite relevant in recent years for handling sequential data, reaching state-of-the-art in many tasks. For this reason and due to the sequential nature of dialogue systems, we implemented a DM using the transformer model for our comparisons.

The implementation is based on the work from (VLASOV; MOSIG; NICHOL, 2019) and Figure 11 shows a schematic representation of the DM architecture using the transformer architecture.

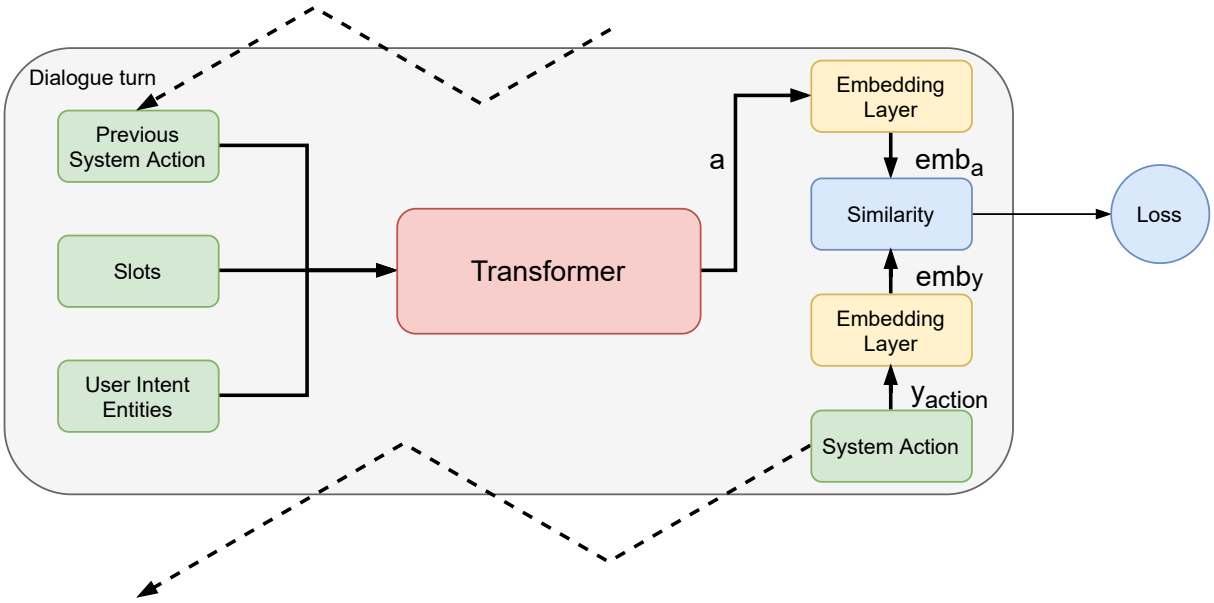


Figure 11: Schematic representation of the transformer dialogue management. Inspired in figure from (VLASOV; MOSIG; NICHOL, 2019)

The input sequence of the transformer architecture is the sequence of the dialogue turns of a conversation. At first, DM encodes the user input, the system action and the slots just as showed before in Section 4.3, resulting in the dialogue state which is the input of the transformer policy. Therefore, the transformer model learns to map the dialogue state into the system action. The attention mechanism embedded in the transformer allows it to look at previous dialogue turns dynamically and use these information to select the best response, i.e., at each turn the transformer attention mechanism computes the relevance of the previous turns that will help the system make its decision. This relevance is what the attention mechanism will learn from data during the training. The transformer output a and the system action y_{action} (which is the label in the supervised approach) are transformed into an embedding through the embedding layer $emb_a = E(a)$, and $emb_y = E(y_{action})$, where $emb \in \mathbb{R}^d$, with $d = 20$ being the embedding dimensions and E represents the embedding layer.

The loss function is based on the similarity between the transformer output embedding and all the system actions $S = emb_a^T emb_y$. Precisely, the loss function for one dialogue is:

$$L_{\text{dialogue}} = -\left\langle S^+ - \log\left(e^{S^+} + \sum_{\Omega^-} e^{S^-}\right)\right\rangle, \tag{4.5}$$

where $S^+ = emb_a^T emb_y^+$, and emb_y^+ is the embedding for the target action y_{action}^+ (true label). $S^- = emb_a^T emb_y^-$, where emb_y^- is the embedding for all other actions y_{action}^- (negative samples) which are elements of the set Ω^-. The idea of this loss is to increase the

similarity between the transformer output and the target, and decrease the similarity between the output and all other actions. Finally the global loss is computed as an average of the loss of all dialogues in the train dataset.

The feed forward layers inside the transformer contain one layer with 128 neurons. The multi-head attention comprises $h = 4$ heads and the dimension size of embedding vectors is $d = 20$. The batch size increases linearly from 8 to 32 for each epoch. So during the training, we sample the batch size of dialogues from the training dataset and for each dialogue we compute the loss, and then the global loss to optimize the transformer with the Adam Optimizer.

During the inference time, the system chooses the action which is the most similar to the transformer output as its response.

In summary, the proposed DCDA-S2M architecture comprises the controller, the slot sharing mechanism and several agents specialized in a single domain. The architecture of each agent is independent and may vary. We proposed the DQN-DM which uses MDP, DQN, softmax and no-flush memory and the DRPO-DM which uses the POMDP, DRQN, softmax and no-flush memory. In the next chapter we show the relevance of each component in these architectures and also make a comparison of the DQN-DM with a supervised learning approach DM based on the transformer architecture. We also evaluate the benefits of DCDA-S2M architecture by comparing it with other RL algorithms and the centralized architecture.

5 EXPERIMENTS AND RESULTS

The major metrics that are analyzed for DM in task-oriented chatbots are: success rate, average reward received and average number of turns (LI; CHEN; LI, 2017; MO et al., 2018; WEISZ et al., 2018). The first refers to the percentage of dialogues in which the agent is able to complete the user's task. The second is a traditional reinforcement learning metric and indicates the average reward the agent received during interactions. Ultimately, the average number of turns gives an idea of the average size of each dialogue. Given these metrics, we can say that a good policy has a high success rate, a high average reward received and a low average number of turns. Due to the strong relationship among these metrics, we evaluated the three of them only in the first set of experiments, for the other experiments, we just presented the success rate metric, which turns to be the most important metric.

In the rest of this chapter, we present the domains used in this work and the experiments executed to obtain the results of the effects of the softmax policy, the no-memory flush strategy and the DRPO-DM method for single domain dialogue system. And finally we present the result of our DCDA-S2M architecture and a comparison between RL and SL approaches using toy (Multiwoz) and real (digital assistant) datasets.

5.1 Domains

There were three domains available for the experiments: movie ticket, MultiWOZ (BUDZIANOWSKI et al., 2018; ERIC et al., 2019) and Itaú Bank's virtual assistant.

The first one is a single domain built with toy datasets and is publicly available [1] so that experiments are reproducible. Its goal is to book a movie ticket that satisfies the user constraints such as movie name, ticket price, date, theater and so on. This domain is used in the baseline (LI; CHEN; LI, 2017) and we used it in experiments regarding single-domain agents using reinforcement learning.

[1] ⟨https://github.com/MiuLab/TC-Bot/tree/master/src/deep_dialog/data⟩

The MultiWOZ domain, is also publicly available[2], but it is multi-domain. More precisely, it contains seven domains: attraction, hospital, hotel, police, restaurant, taxi, and train. So, the agent's task is to accomplish the user goal in one or more domains in a single conversation. For example, the user may want to go to an expensive Japanese restaurant at 08:00pm and also book a five-star hotel, in the north area at 10:00pm.

The Itaú Bank's domain comes from real-world data and this data is private due to bank's privacy concerns. All the data follow the GDPL (General Data Protection Law), in which sensible data were not employed and client's information was anonymised. This domain is useful to illustrate the relative performances of the assessed methods in more complex domains used in the real-world. The goal of this domain is to solve some client's interactions related to the bank. For example, in this domain, a rule is composed of the intent `doubt` and the slots `credit card` and `password`; then the corresponding action would be related to solve a problem with the credit card's password. In this work, the issues are specifically related to non-account holders, such as problems with credit card and passwords, as well as doubts about loans.

Table 4 shows some examples of the ontology (intent/slot/data) for each domain, including their respective statistics. Here the data are basically examples of interactions presented in the dialogues of each domain. Note that the third domain, which comes from a real-world application, is more complex than the other two due to the higher number of intents and slots. The data in each domain are splitted such that 80% of them are used for training and the remaining 20% are used for testing when training with supervised learning.

Conversations between the agent and the user have similar structures for all domains. During the interactions, the user informs some slots that are in his/her goal and the agent evaluates if it is enough to complete the task. Otherwise, it just waits for the user to inform something else or request for some additional information. Figure 12 shows examples of the interactions between the user and the agent in all domains and their respective translations to dialogue acts.

5.2 Softmax

In this section, the agent's performance is analyzed with the proposed softmax policy for balancing between exploration and exploitation, described in Section 4.3.1. We com-

[2] ⟨https://github.com/budzianowski/multiwoz⟩

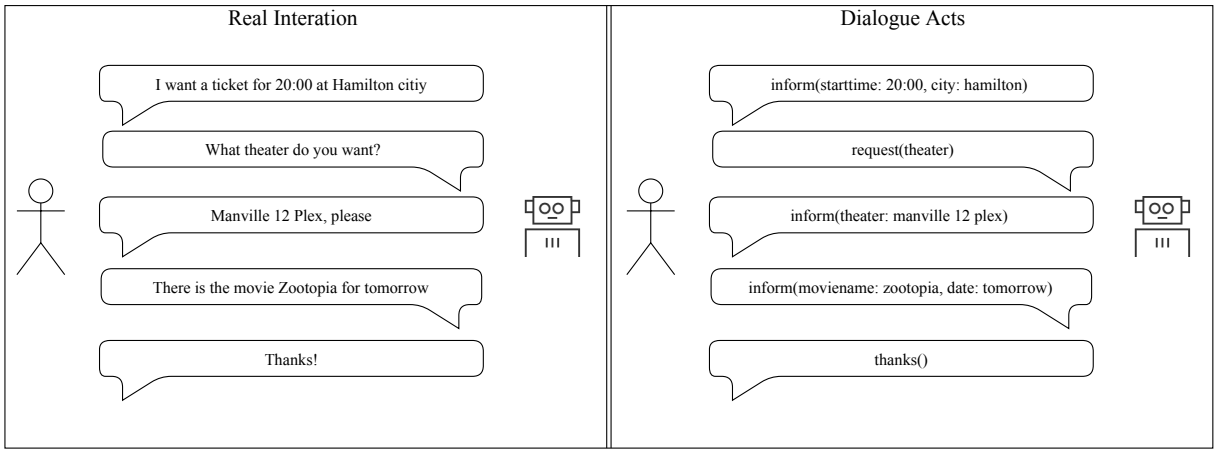
Table 4: Some examples of the ontology for each studied domain.

Domain	**Movie Ticket**
Intent	inform, request, thanks, greeting, $\cdots$ (Total: 6)
Slot	moviename, starttime, data, genre, theater, numberofpeople, $\cdots$ (Total: 20)
Data	inform(starttime: 20:00, theater: manville 12 plex) $\rightarrow$ inform(moviename: zootopia) (Total: 991)
Domain	**MultiWoz**
Intent	welcome, inform, request, select, not found, bye, $\cdots$ (Total: 13)
Slot	aaddress, area, pricerange, name, phone, day, $\cdots$ (Total: 24)
Data	restaurant-inform(area: north, pricerange: moderate) $\rightarrow$ restaurant-inform(name: The Nirala) (Total: 1482)
Domain	**Digital Assistant**
Intent	doubt+invest, consult, negotiate, $\cdots$ (Total: dozens)
Slot	product, status, service, people, category, channel, functionality, $\cdots$ (Total: hundreds)
Data	consult(product: credit card) $\rightarrow$ inform(debt: X) (Total: thousands)

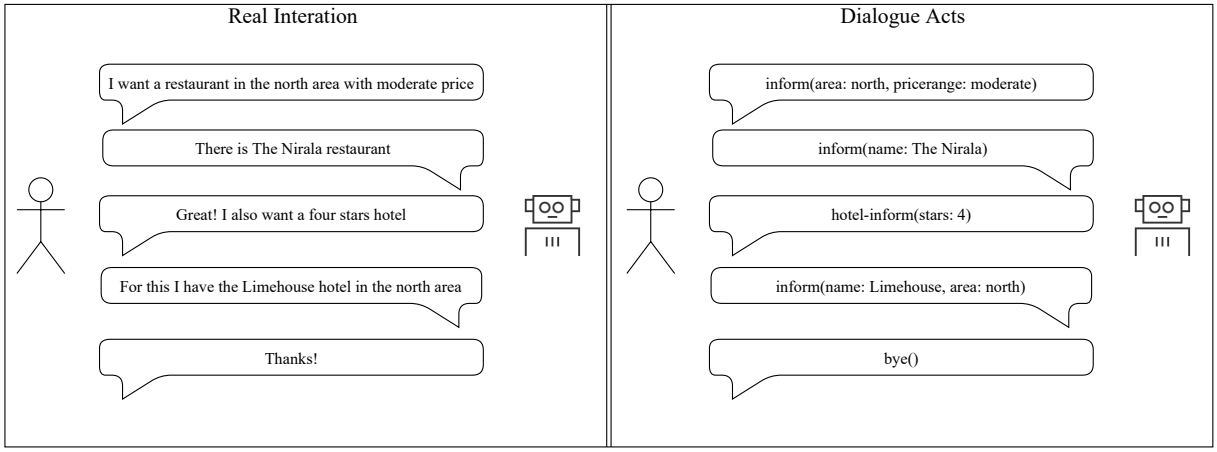
pare the results with the ϵ-greedy policy adopted in the baseline work (LI; CHEN; LI, 2017) in the movie ticket domain. The experiments were repeated 20 times in order to construct the learning curve graphs with a confidence interval of 95% (Figure 13).

There are three points that can be analyzed to compare two reinforcement learning agents: initial jump, time to a certain limit and asymptotic performance. The initial jump represents the improvement in performance in the first episode, the time to a certain limit indicates how many episodes it took to reach a certain limit (usually 70% maximum) and asymptotic performance is the performance achieved by the agent after learning stabilize.

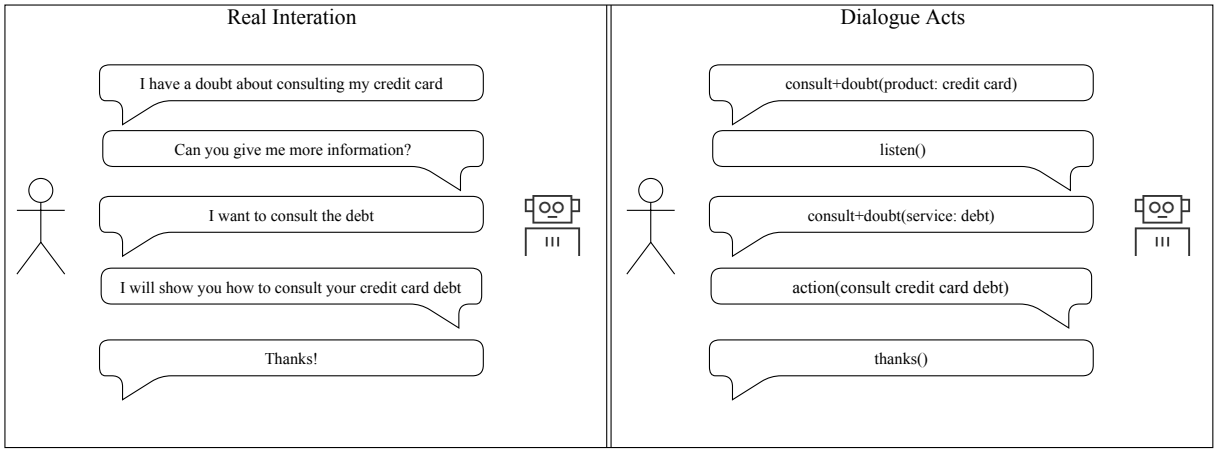
From the analysis of the graphs of Figure 13, it is possible to observe that the proposed method shows a statistically significant improvement over the base model during the learning phase, in the three observed items. To get an idea of the improvement, looking at the success rate curve, there is an improvement in the initial jump of approximately 17.24%, the time difference to reach the 0.5761 limit (which is 70% of the maximum amount) is 15 times and the improvement in asymptotic performance is about 8.33%.



(a) Movie Ticket



(b) MultiWOZ



(c) Digital Assistant

Figure 12: Example of real interactions (left side) between the user and the agent and their respective translations to dialogue acts (right side) for (a) movie ticket, (b) MultiWOZ and (c) the digital assistant domain.

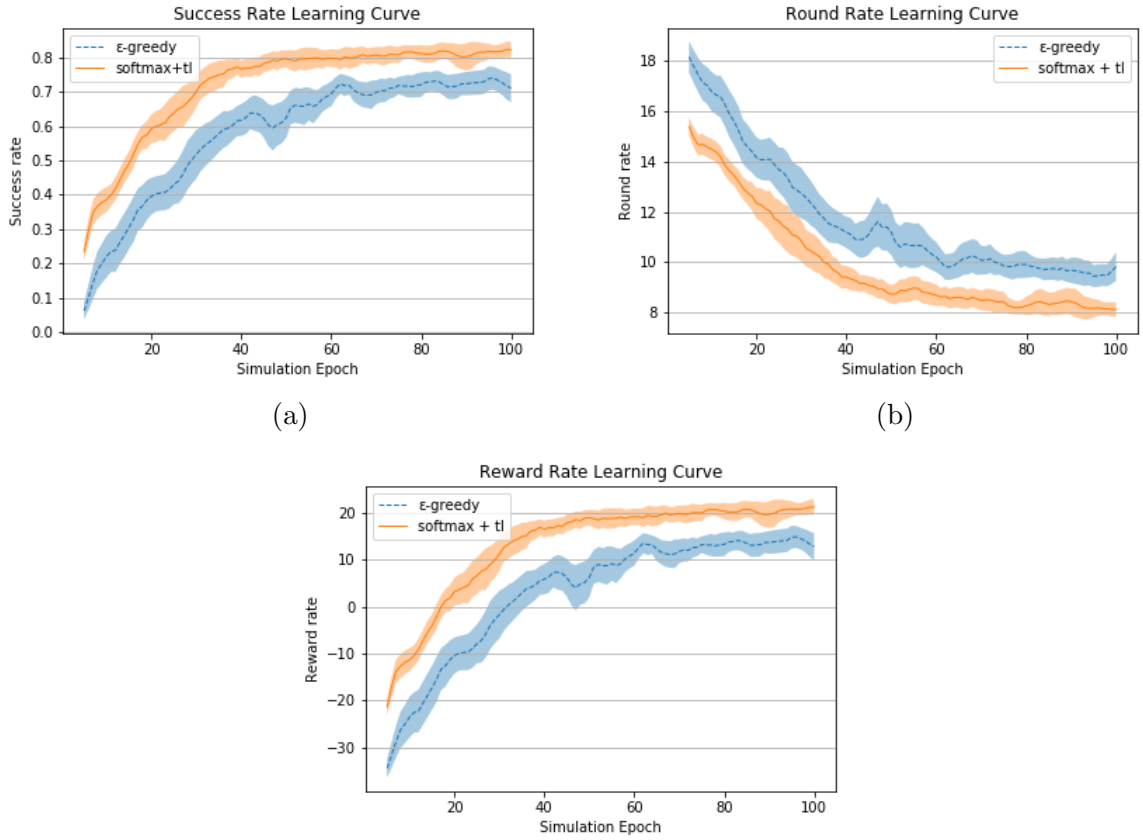
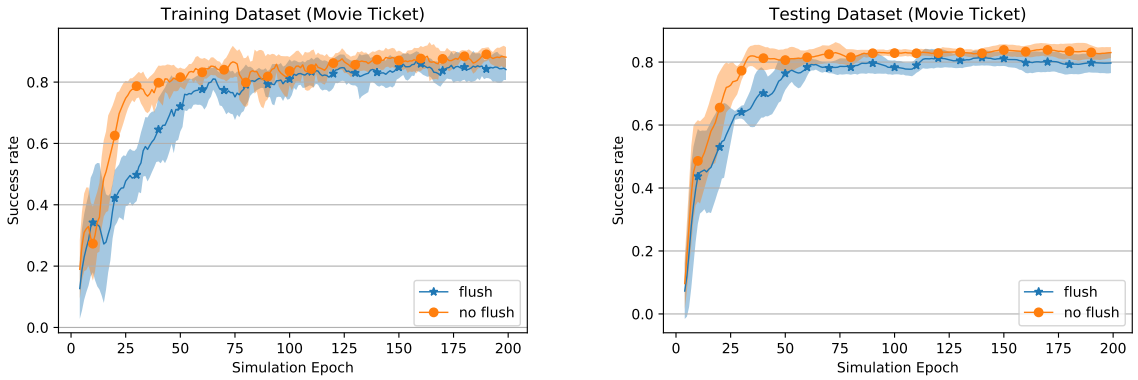


Figure 13: Learning curve for the (a) Success rate, (b) Average round, and (c) Average reward. Source: (NISHIMOTO; REALI COSTA, 2019)

5.3 Memory Flush

Figure 14a shows the learning curve of the success rate during training for the movie ticket domain. We compare our proposed approach, which involves not flushing the replay memory, with the previous works (LI; CHEN; LI, 2017; NISHIMOTO; REALI COSTA, 2019) that flush the replay memory. Although both procedures achieve comparable accuracy, the no-flush approach causes the agent to learn faster and achieves the asymptotic performance before the flush method. For this specific experiment, we only evaluated the results on the movie ticket domain as there already was an established baseline to compare.

While our proposal reaches an average success rate greater than 0.8 at epoch 42 on training dataset, the flush method only reaches such a result at epoch 82. We can view this improvement in the learning speed as a better usage of the designer knowledge. Indeed, if we flush the memory replay after some threshold, all the rule actions took in the warm-up phase are lost. Consequently, not flushing the memory keeps the designer knowledge in



(a) (b)

Figure 14: Learning curve for the movie ticket domain on the a) training and b) testing dataset by flushing and not flushing (our approach) the experience replay memory.

memory replay as long as possible and makes a better use of it.

Therefore, we can state that not flushing the replay memory improves the agent's knowledge generalization regarding the warm-up rules, thus leading the agent to learn an appropriate policy faster, as shown in Figure 14b.

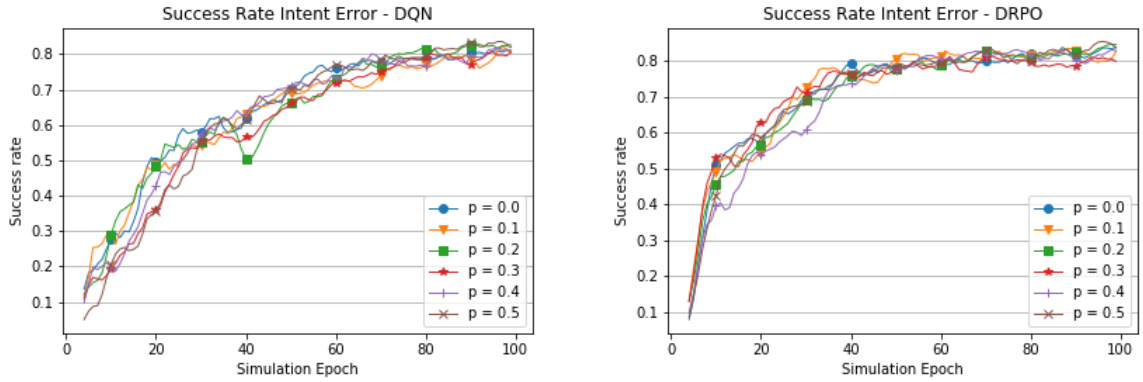
5.4 Dialogue Management with DRPO

The agent was also trained using the POMDP framework described in Section 4.3.3, named DRPO-DM. In this case, we intend to analyze the robustness of the DM policies regarding errors from the NLU. For these experiments, the runs were repeated 10 times generating curves with a confidence interval of 95%. First we analyze the impact of errors in the classification of intention and second, the impact of errors in the extraction of slots. The levels of errors, i.e., the probability of an error occurring, ranged between 0.0 and 0.5.

Figure 15 shows the impact of errors in the classification of intention in both algorithms, DQN (with MDP modeling) and DRPO (with POMDP modeling). Figure 16 shows the same analysis for errors in the extraction of slots. Finally, Figure 17 compares both models in a specific probability error (0.3).

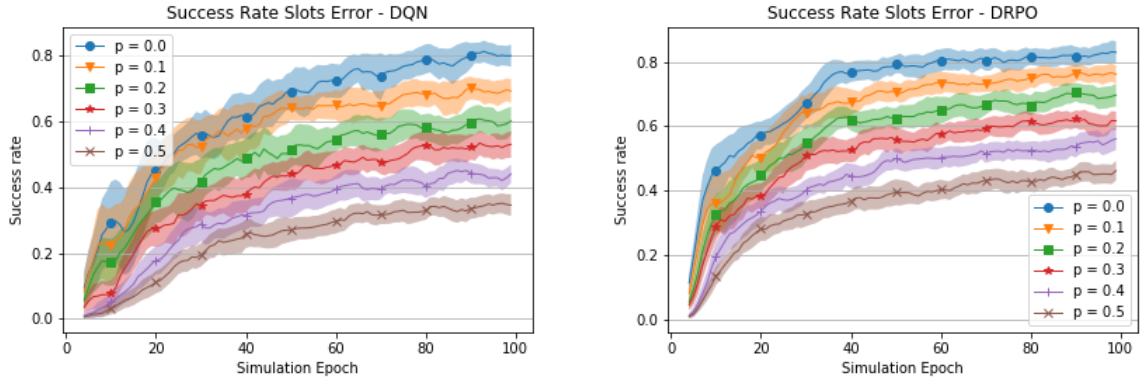
It is possible to verify that the error in the classification of the intent did not impact the agent's performance. This specific graph was not plotted with the confidence interval just to allow a clearer visualization.

On the other hand, it is possible to verify that the errors in the extraction of the



(a) (b)

Figure 15: Success rate for different levels of intent errors. (a) for DQN algorithm. (b): for DRPO algorithm. Source: (NISHIMOTO; REALI COSTA, 2020)



(a) (b)

Figure 16: Success rate for different levels of slot errors. (a) for DQN algorithm. (b): for DRPO algorithm. Source: (NISHIMOTO; REALI COSTA, 2020)

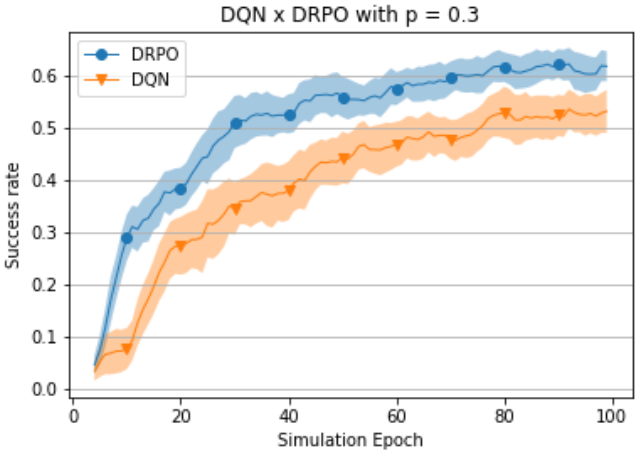


Figure 17: DRPO x DQN with p=0.3 error rate.

slots of the sentence cause a significant decay in the agent's performance. And this is

expected, since there is no easy "tip" to suggest that there was an error. For example, if the user informs `moviename: ‘‘The Avengers’’` but the NLU understood `moviename: ‘‘Joker"`, there is nothing to indicate that "Joker" is possibly wrong. However, it is possible to observe that the DRPO algorithm, which uses POMDP, presents a smaller decrease in performance with an increase in the level of error. Thus, it can be said that the use of POMDP makes the system more robust.

Moreover, comparing the two models with 0.3 probability of error, as seen in Figure 17 the improvement achieved by DRPO-DM is statistically significant (as the curves were plotted with 95% confidence interval) in both, the asymptotic performance and the time to threshold. The asymptotic performance is the success rate that the agent stabilizes while time to threshold means the number of epochs the curve achieves a specific success rate.

5.5 Reinforcement Learning vs Supervised Learning

The results of the comparison between the supervised and reinforcement learning approaches for both the MultiWOZ dataset and the virtual assistant domain are shown in Tables 5 and 6, respectively. They show the results with 95% confidence interval in the testing phase. We can see that although they present similar performance in the accuracy, the reinforcement learning approach shows better results on the other metrics. It means that the agent trained with reinforcement learning can learn better on how to fulfill the user goal slots during the dialogue. The intuition behind this is that the sequential nature of reinforcement learning algorithms fits well in the dialogue management problems. Even though the transformer networks works well with sequential data, it does not leverage the dynamic interactions that we have in reinforcement learning.

We can also see that this behaviour (RL with better performance than SL) keeps in the virtual assistant domain. It was expected a decreasing in the performance compared to the MultiWOZ domain because of the domain complexity (greater number of slots, actions and intents) and the difficulty of collecting good data for training.

Table 5: Comparison between the SL and RL approaches in the MultiWOZ domain. Metrics with 95% confidence interval in the test dataset.

MultiWOZ	F1	Precision	Recall	Accuracy
Transformer	0.61 ± 0.02	0.63 ± 0.01	0.61 ± 0.04	0.70 ± 0.01
DQN	0.83 ± 0.04	0.78 ± 0.06	0.89 ± 0.08	0.71 ± 0.04

Table 6: Comparison between the SL and RL approaches in the virtual assistant domain. Metrics with 95% confidence interval in the test dataset.

Virtual Assistant	F1	Precision	Recall	Accuracy
Transformer	0.57 ± 0.01	0.57 ± 0.01	0.59 ± 0.04	0.73 ± 0.01
DQN	0.79 ± 0.04	0.78 ± 0.06	0.78 ± 0.03	0.78 ± 0.04

5.6 Slot Similarity

For visualization of the learned node embedding in the MultiWOZ domain we used a t-distributed stochastic neighbor embedding (t-SNE) model with perplexity 5 using the scalar product as similarity function. Figure 18 shows this visualization.

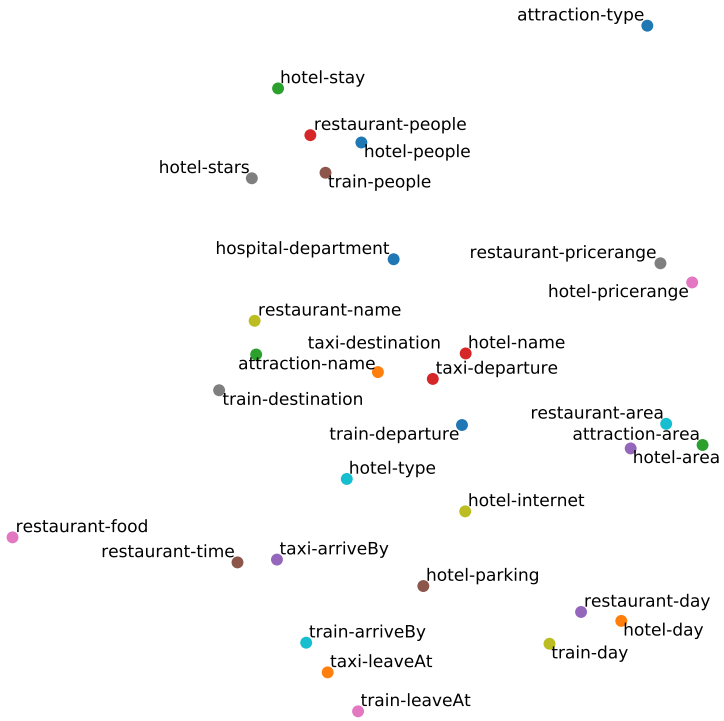


Figure 18: Visual representation of the learned node embedding.

Figure 18 clearly shows some groups of nodes that are related to each other. For example, *hotel-area*, *attraction-area*, and *restaurant-area* forms a group, indicating that users generally request places in the same area. It also happens for the price range (*restaurant-pricerange* and *hotel-pricerange*), day (*restaurant-day*, *hotel-day*, and *train-day*) and people (*restaurant-people*, *hotel-people*, and *train-people*) slots. Although *hotel-stay* and *hotel-stars* looks close to the group with slot "people", computing their similarity with *restaurant-people* we got 0.118 and 0.088, respectively – thus they are not similar and should not share values. On the other hand, the similarity between *restaurant-area* and *attraction-area* is 0.91 showing that they are similar and must share their values inside a

conversation. Here we used a similarity of 0.8 as a threshold for sharing the slots values.

An interesting observation is that *attraction-name* and *hotel-name* are quite close to *taxi-departure*, with similarity 0.57 and 0.668, respectively, but they are not close to each other, i.e., the similarity between them is 0.011. This is expected since it is not common an attraction with the same name as a hotel.

5.7 DCDA-S2M Evaluation

To evaluate our proposal we assessed four models: the baseline Rule-based policy available in ConvLab-2, the VMLE[3] (Vanilla Maximum Likelihood Estimation) policy , and the PPO algorithm trained in both approaches: a centralized system with a unique agent trained to handle all domains at once using PPO and DQN (PPO_{all} and DQN_{all}, respectively) and our proposal with the single agents trained with both PPO and DQN (DCDA_{ppo} and DCDA_{dqn}, respectively). Table 7 summarizes the models that were compared.

Table 7: Models compared.

Model	*Description*
Rule	ConvLab-2 Rule-based policy
VLME	Policy obtained from the VLME algorithm
$\textbf{PPO}_{all}$	A single-agent centralized PPO DM trained to handle all domains
$\textbf{DQN}_{all}$	A single-agent centralized DQN DM trained to handle all domains
$\textbf{DCDA}_{ppo}$	DCDA with PPO-trained individual agents
$\textbf{DCDA}_{dqn}$	DCDA with DQN-trained individual agents

The metrics are automatically computed by the evaluator presented in ConvLab-2 and encompasses the complete rate, success rate, book rate, precision, recall, F1-score for the informed slots, and average number of turns for both the dialogues that were successful and the total set of dialogues. The complete rate indicates the rate of dialogues that could finish (either with success or fail) before achieving the maximum number of turns. The precision, recall and F1-score indicate the ability of the agent to fulfill the slots of the user goal, i.e., leads to the correct slot. Tests were performed over 2000 dialogues in the MultiWOZ domain.

Table 8 shows the evaluation results for all the six models in the pipeline setting, i.e., without the NLU and NLG modules. As expected the rule policy performs almost

[3]VMLE is the simple maximum likelihood estimation which employs a multi-class classification via imitation learning

"perfectly" succeeding in 98.45% of the dialogues and it can serve as a baseline. This is so because the rule policy is designed specifically for this domain and knowing how the simulator works (the disadvantage is that it does not work in other domains). Among the trainable agents, the multiple agents models (DCDA) perform better than a single agent in almost all aspects achieving more than 88% of success rate and more than 94% of complete rate in both DCDA_{ppo} and DCDA_{dqn}, while PPO_{all} and DQN_{all} present success rate lower than 75%. This shows a much better performance and efficiency as it can solve more user tasks using less number of turns. The average number of turns in all dialogues is 14.92 and 12.99 for DCDA_{ppo} and DCDA_{dqn}, respectively, which is very close to the baseline rule policy (13.48) showing it could learn a very good policy in solving tasks.

The increase in the average number of turns for all dialogues can be explained by analysing the failed dialogues during test. It can be seen that in many failed dialogues the conversation went in a loop with the agent and the simulated user, repeating the same act of dialogue consecutively. In these cases the dialogue lasts the maximum number of turns, contributing to the increase in the average number of turns. Figure 19 shows an example of a failed dialogue in which the user and the agent went in a loop. The reason why this phenomenon occurs is not very clear to us. The worst performance of the VLME is expected, as the other agents depend on the VLME for pre-training and it is a simple algorithm that employs imitation learning.

Restaurant	
	. . .
	usr: [Restaurant, Inform, Price, expensive]
	agt: [Restaurant, Inform, Name, curry garden], [Restaurant, Inform, Addr, 106 Regent Street City Centre]
	usr: [Restaurant, Inform, Area, south]
	agt: [Restaurant, Inform, Name, he good luck chinese food takeaway], [Restaurant, Inform , Addr, 82 Cherry Hinton Road Cherry Hinton]
	usr: [Restaurant, Request, Food, ?]
	agt: [Restaurant, Inform, Name, he good luck chinese food takeaway], [Restaurant, Inform , Addr, 82 Cherry Hinton Road Cherry Hinton]
	usr: [Restaurant, Request, Food, ?]
	. . .

Figure 19: Example of a fail dialogue where the agent went in a loop. The user keeps requesting the food type and the agent keeps informing the restaurant name and address.

We can also see that our DQN algorithm - with the improvements of the softmax strategy and no-flush memory replay - shows slightly better results than PPO in most metrics. It is only worst in precision and consequently in F1-Score. The reason for this is that the agent trained with DQN "guesses" more than the agent trained with PPO, that is, the agent tries to inform some slots that the user has not requested yet based on the available options and the constraints given by the user.

Table 8: Results for agents tested in a pipeline setting. The best results are in bold.

	Rule	**VLME**	**PPO**$_{all}$	**DQN**$_{all}$	**DCDA**$_{ppo}$	**DCDA**$_{dqn}$
Success	98.45	39.57	66.63	71.2	88.14	**89.68**
Complete	98.45	41.50	77.17	89.62	94.01	**94.78**
Book	98.79	1.35	60.20	90.41	88.01	**96.21**
Precision	83.47	65.24	77.57	58.16	**80.26**	74.79
Recall	99.16	68.82	86.53	87.84	97.02	**97.68**
F1	88.55	64.12	79.12	67.45	**86.00**	80.72
Turn (suc)	13.40	13.80	13.62	11.93	13.84	**11.34**
Turns (all)	13.48	22.39	19.82	15.66	14.92	**12.99**

We also evaluated the effects of using or not the S2M in the rule and DCDA agents. The results of this experiment regarding the use of the slot sharing mechanism are presented in Table 9. Results show that for the Rule policy the sharing mechanism also helped the agent to have a slightly better performance. Although the success rate for DCDA did not change much, the sharing mechanism also helped it to have a better complete and book rate. Another enhancement was in the average number of turns. The average number of turns required in successful dialogues for the Rule and DCDA$_{ppo}$ policies decreased from 13.40 to 13.20, and from 13.84 to 13.43, respectively, when the sharing mechanism was incorporated. Thus we can see that the sharing mechanism makes the conversational agent complete dialogues faster than it would without this mechanism. The DQN algorithm did not show improvement in turns with the S2M mechanism.

Table 9: Evaluation of the use of S2M in the Rule policy and DCDA, with the goal generator generating random goals. Best results are in bold.

	Rule		**DCDA**$_{ppo}$		**DCDA**$_{dqn}$	
	with S2M	**no S2M**	**with S2M**	**no S2M**	**with S2M**	**no S2M**
Success	**98.60**	98.45	**88.24**	88.14	**89.84**	89.68
Complete	**98.65**	98.45	**95.94**	94.01	**94.99**	94.78
Book	**99.25**	98.79	**90.50**	88.01	**96.49**	96.21
Precision	83.32	**83.46**	**80.31**	80.26	72.34	**74.79**
Recall	99.14	**99.16**	**97.04**	97.02	**97.87**	97.68
F1	88.46	**88.55**	**86.02**	85.99	**80.80**	80.72
Turn (suc)	**13.20**	13.40	**13.43**	13.84	11.39	**11.34**
Turns (all)	**13.23**	13.48	**14.77**	14.92	**12.99**	**12.99**

An interesting fact is that besides the slightly better performance with the sharing mechanism, the precision, recall and F1-score did not followed the same behavior, i.e., they had better results or very close (less than 0.05%) results as those without the sharing mechanism. This result is not very surprising because as the agent with the sharing mechanism tries to "guess" the slots of new domains within the conversation, it ends up

reporting more wrong slots of the user goal causing worse precision, recall, and F1-score.

All theses experiments were assessed with the user simulator generating random goals based on a distribution of the goal model extracted from the dataset. So this can include simple goals within only one domain or goals that span to more than one domain but do not have any slot with the same value to share. Indeed, among all 2000 goals generated during testing, only about 400 contain common values between slots. With that in mind, we ran another test of the sharing mechanism that restricts the user simulator to only generating goals that contain common slots. Therefore, the generated goals end up being more complex in general than those generated in the first test.

Table 10 shows the results. There is an expected significant decrease in the general performance due to the increase in user goals complexity. However, here we can clearly observe the great advantage of the S2M sharing mechanism in this setting.

Table 10: Evaluation of the use of S2M in the Rule policy and DCDA with the goal generator generating slots with common values. Best results are in bold.

	Rule		**DCDA$_{ppo}$**		**DCDA$_{dqn}$**	
	with S2M	**no S2M**	**with S2M**	**no S2M**	**with S2M**	**no S2M**
Success	**92.60**	80.35	**78.41**	68.42	**78.80**	72.08
Complete	**92.50**	80.45	**92.13**	85.49	**90.19**	87.73
Book	**96.11**	95.88	**83.58**	83.10	**91.12**	89.85
Precision	**79.89**	79.34	**76.82**	74.88	**65.53**	64.40
Recall	**95.43**	86.80	**95.15**	90.01	**95.55**	92.63
F1	**83.99**	77.38	**82.92**	78.77	**75.23**	73.02
Turn (suc)	**17.15**	18.24	**19.97**	20.69	**15.43**	15.92
Turns (all)	**17.10**	17.60	**21.28**	21.57	**17.89**	18.03

There is a 12.25%, 9.99% and 6.72% success rate difference with the Rule and DCDA policies, respectively. We also see a bigger impact on the average number of turns. It affects mostly the successful dialogues because the number of turns is affected only when the transferred slots values are correct – otherwise the user would still need to inform these slots – and chances of a successful dialogue increase when it happens. Finally, we also see a better precision, recall and F1-score for the agent with the sharing mechanism. Since all goals in this tests have at least one common value among the slots, the agent "guesses" are more likely to be correct.

Figures 20 and 21 show examples of system generated dialogues using and not using the slot sharing mechanism, respectively.

Observe that when the domain switched to the hotel domain, the agent in Figure 20 asked for confirmation if the price is moderate and area is north and recommended a hotel

Goal	**restaurant**: info: {area: north, food: indian, pricerange: moderate}, reqt: {postcode: ?, address: ?}, **hotel**: info: {parking: yes, pricerange: moderate, stars: 4, area: north}, reqt: {type: ?}}
Restaurant	. . .
	usr: [Restaurant, Inform, Price, moderate], [Restaurant, Inform, Area, north]
	agt: [Restaurant, Inform, Name, the nirala]
	usr: [Restaurant, Request, Addr, ?], [Restaurant, Request, Post, ?]
	agt: [Restaurant, Inform, Addr, 7 Milton ....], [Restaurant, Inform , Post, cb41uy]
Hotel	**usr:** [Inform, Hotel, Starts, 4
	agt: [Confirm, Hotel, Price, moderate], [Confirm, Hotel, Area, north], [Recommend, Hotel, Name, limehouse],
	usr: [Inform, Hotel, Parking, yes]
	agt: [Inform, Hotel, Choice, 6], [Recommend, Hotel, Name, kirkwood house]
	usr: [Request, Hotel, Type, ?]
	agt: ['Inform', 'Hotel', 'Type', 'guesthouse']
	. . .

Figure 20: Example of dialogue using the slot sharing mechanism, resulting in a dialogue length 8.

Goal	**restaurant**: info: {area: north, food: indian, pricerange: moderate}, reqt: {postcode: ?, address: ?}, **hotel**: info: {parking: yes, pricerange: moderate, stars: 4, area: north}, reqt: {type: ?}}
Restaurant	. . .
	usr: [Restaurant, Inform, Price, moderate], [Restaurant, Inform, Area, north]
	agt: [Restaurant, Inform, Name, the nirala]
	usr: [Restaurant, Request, Addr, ?], [Restaurant, Request, Post, ?]
	agt: [Restaurant, Inform, Addr, 7 Milton ....], [Restaurant, Inform , Post, cb41uy]
Hotel	**usr:** [Inform, Hotel, Starts, 4]
	agt: ['Request', 'Hotel', 'Area', '?']
	usr: ['Inform', 'Hotel', 'Area', 'north']
	agt: ['Request', 'Hotel', 'Price', '?']
	usr: ['Inform', 'Hotel', 'Price', 'moderate']
	agt: [Recommend, Hotel, Name, limehouse]
	. . .

Figure 21: Example of dialogue that does not use slot sharing mechanism, resulting in a dialogue length 11.

with these constraints. In natural language we could think in this dialogue act as: "Do you want a hotel in north with a moderate price, right? There is the hotel Limehouse". In this way, the user did not need to inform these slots again, saving some turns until task completion. While in Figure 21 the agent needed to ask again the area and price for the user, resulting in a redundant dialogue which takes more turns to be completed (11 turns against 8 turns).

One drawback for the DCDA-S2M is the training time required for training all the agents. Table 11 shows the average training time for each agent. As we can see, the total amount of time required to train all seven agents (attraction, hospital, hotel, police, restaurant, taxi, train) is 291.29 minutes, which is approximately 16% more than centralized system training (188.97). However, it is worth noting that agents could be trained in parallel, which would require greater computational power.

Table 11: Training time in minutes for each agent. The hardware used in these experiments was a graphic card NVIDIA GeForce GTX1060 and processor Intel Core i7-7700HQ.

	Centralized	Attraction	Hospital	Hotel	Police	Restaurant	Taxi	Train
Time to Train (min)	188.97	32.01	28.87	29.97	26.85	32.01	29.77	111.81

5.8 Discussion

The results showed that the main goal of this work, the use of a distributed architecture with multiple agents trained separately for each domain, can leverage the system performance compared to the same algorithm used to train a single agent for all domains at once. This is because each agent can specialize in solving its own problem well, which is much simpler than solving tasks well in all domains, as with the centralized approach in a single agent. Furthermore, distributed systems can add new domains without the need to retrain the entire system.

The use of the slot sharing mechanism also proved to enhance system performance, especially for tasks where the goal has some common slot across domains. Besides improving the system's success rate, it also decreases the average number of turns, showing that the system avoided asking for redundant information.

A major disadvantage of DCDA-S2M is the need to train several agents separately and this can be time and energy consuming. In this sense, for future work we intend to explore transfer learning techniques in reinforcement learning to accelerate the training of new agents.

We also made several experiments regarding the reinforcement learning algorithms applied to single domain systems. The first set of experiments showed that a good balance between exploration and exploitation can considerably improve the agent's performance. In particular that the softmax strategy outperforms the basic ϵ-greedy one. Indeed, with the ϵ-greedy strategy, when exploring, all actions have the same probability to be chosen, that is, the agent does not use any previous information to favor one action over another. In other words, bad actions can be equally chosen than good actions, and this can delay the agent's learning. On the other hand, with softmax policy, the agent samples an action following a softmax distribution based on Q-values of each action. Thus, in the beginning of the learning, when the agent knows nothing about each action and consequently the Q-values are similar, it tends to explore more. As the agent refines its estimations over the time, bad actions become less probable to be chosen but always with a possibility to explore. As the results showed, it improve the performance of the agent.

The next experiments showed how not flushing the replay memory helps the agent to learn faster. The primary reason for this to happen is that during the first episodes of the learning phase, the replay memory is still filled with some experiences derived from pre-defined rules, in which are inherited the designer knowledge. This means that if we

flush the memory too early, we would lose these experiences and not make the most of it. Thus, it is important to keep them enough time until the agent acquires useful learning. Since the replay memory has limited size, newer experiences will eventually replace them, avoiding the agent to be biased by the rules.

The experiments regarding POMDP framework showed that `intent` errors does not affect significantly the success rate of our agent. One possible explanation is that, since the intent depends a lot on context, the agent can infer the correct intent, even if it got it wrong. For example, if the agent's last action was to request something, then it is quite likely that the user's action is to inform something. On the other hand, it does not happen for the `slots` errors. This means that the agent has a harder time detecting slot errors and eventually keeps the conversation going even with the wrong slot. The intuition for this is that it is difficult to detect if a slot is misclassified, for example, if the user informs `moviename: "Avengers"` but DM receives `moviename: "Joker"` there is no clue that this is wrong. However, results presents that DRPO obtains better results than DQN. This could be due to the recurring layer in the network that handles the sequential nature of the dialogue and retains some historical information to help detect these errors.

In the further experiments, we observed that RL presents better results than SL on test dataset for both domains, MultiVOZ and the bank virtual assistant. A potential explanation for this phenomenon resides in the low amount of rules used to train the supervised agent. This lack of available data for specific domains is a common issue for task-oriented chatbots. It shows that the supervised agent is more susceptible to overfit due to its lower performance on unseen data, i.e., although the SL approaches achieve a good performance on training dataset, it could not generalize well the knowledge learned and got bad performance on testing dataset. This behaviour is expected since RL techniques have the capability of exploring the environment and thus can find an optimal policy with less data. Therefore, as reinforcement learning can handle better with the lack of data, it presents better results. Furthermore, because of the sequential decision making nature of RL, the agent can keep track of the history of the interactions with the state representation and thus it has more information to choose an action.

Finally, with these experiments so far, we have made some improvements on DQN algorithm for dialogue management and concluded that modular RL approaches presents better results than simple SL modular approach. Furthermore, we also validated this comparison on a real-world complex dataset.

6 CONCLUSIONS AND FUTURE WORK

In this work, we showed that the slot sharing mechanism enhances the system performance, specially for tasks where the goal contains some common slots across domains. In these scenarios, the system takes advantage of the S2M to transfer knowledge across domains and complete the dialogue successfully. Besides improving the success rate, it also decreases the average number of turns as the system avoids asking redundant information in the same dialogue.

Furthermore, the divide-and-conquer approach for multi-domain dialogue systems proved to achieve good results using basic reinforcement learning algorithms. This is because each agent can specialize in solving its own problem which is much simpler than the whole one. The distribute architecture is also good for scaling in the number of domains since we only need to add a new trained agent instead of retraining the entire system. Regarding the S2M, it would not know how to transfer slots from/to the new domain until we have enough data to learn the node embedding. However it's not a big problem since we can use the S2M to transfer slots between old domains while collecting data to learn the embeddings of the new domain.

In addition to these contributions for multi-domain dialogue systems, we also have improvements for single domain task-oriented dialogue systems, specifically on dialogue management for those with a pipeline architecture. Using a movie ticket dataset and the work of Li, Chen and Li (2017) as baseline we handled three important issues: a better balance between exploration and exploitation for efficient learning (NISHIMOTO; REALI COSTA, 2019); an enhancement on the use of expert knowledge to speed up the agent's initial learning; and a method to handle with noises and errors coming from NLU (NISHIMOTO; REALI COSTA, 2020). Finally, we perform a comparison between our RL implementation and a current SL approach, both applied directly in dialog management. The results show that RL-based methods are more promising than SL-based ones.

For future we can explore the other components of the dialogue system (NLU and NLG) in order to have a complete system. Another promising path is to explore the trans-

fer learning techniques (ZHU KAIXIANG LIN; ZHOU, 2020) to accelerate the training of new agents.

REFERENCES

ACHIAM, J. Spinning Up in Deep Reinforcement Learning. 2018.

AHMED, A. et al. Distributed large-scale natural graph factorization. In: *Proceedings of the 22nd International Conference on World Wide Web*. [S.l.: s.n.], 2013. p. 37–48. ISBN 9781450320351.

BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. In: BENGIO, Y.; LECUN, Y. (Ed.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [s.n.], 2015. Disponível em: ⟨http://arxiv.org/abs/1409.0473⟩.

BELLMAN, R. A Markovian Decision Process. *Indiana Univ. Math. J*, v. 6, p. 679–684, 1957. ISSN 0022-2518.

BOCKLISCH, T. et al. Rasa: Open Source Language Understanding and Dialogue Management. *ArXiv*, abs/1712.05181, 2017.

BORDES, A.; WESTON, J. Learning End-to-End Goal-Oriented Dialog. In: *International Conference on Learning Representations (ICLR) 2017*. [S.l.: s.n.], 2017.

BUDZIANOWSKI, P. et al. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018. p. 5016–5026. Disponível em: ⟨https://aclanthology.org/D18-1547⟩.

CHEN, H. et al. A survey on dialogue systems: Recent advances and new frontiers. *SIGKDD Explor. Newsl.*, ACM, New York, NY, USA, v. 19, n. 2, p. 25–35, nov. 2017. ISSN 1931-0145. Disponível em: ⟨http://doi.acm.org/10.1145/3166054.3166058⟩.

CHEN, L. et al. AgentGraph: Towards Universal Dialogue Management with Structured Deep Reinforcement Learning. *ArXiv:abs/1905.11259*, 2019.

CHO, K. et al. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1724–1734. Disponível em: ⟨https://aclanthology.org/D14-1179⟩.

CUAYÁHUITL, H.; KEIZER, S.; LEMON, O. Strategic dialogue management via deep reinforcement learning. In: *NIPS'15 Workshop on Deep Reinforcement Learning*. [s.n.], 2015. Disponível em: ⟨http://arxiv.org/abs/1511.08099⟩.

CUAYÁHUITL, H. et al. Evaluation of a hierarchical reinforcement learning spoken dialogue system. *Computer Speech & Language*, v. 24, n. 2, p. 395–429, 2010. ISSN 0885-2308.

CUAYÁHUITL, H. et al. Deep reinforcement learning for multi-domain dialogue systems. *CoRR*, abs/1611.08675, november 2016. Disponível em: ⟨http://arxiv.org/abs/1611.08675⟩.

CUAYÁHUITL, H. et al. Scaling up deep reinforcement learning for multi-domain dialogue systems. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2017. p. 3339–3346.

DAI, Y. et al. A Survey on Dialog Management: Recent Advances and Challenges. *ArXiv:abs/2005.02233*, 2020.

DHINGRA, B. et al. Towards End-to-End Reinforcement Learning of Dialogue Agents for Information Access. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, 2017. p. 484–495. Disponível em: ⟨https://www.aclweb.org/anthology/P17-1045⟩.

DODGE, J. et al. Evaluating Prerequisite Qualities for Learning End-to-End Dialog Systems. In: BENGIO, Y.; LECUN, Y. (Ed.). *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. [S.l.: s.n.], 2016.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, v. 12, n. 61, p. 2121–2159, 2011.

ERIC, M. et al. MultiWOZ 2.1: Multi-Domain Dialogue State Corrections and State Tracking Baselines. *CoRR*, abs/1907.01669, 2019. Disponível em: ⟨http://arxiv.org/abs/1907.01669⟩.

GAO, J.; GALLEY, M.; LI, L. Neural Approaches to Conversational AI. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*. Melbourne, Australia: Association for Computational Linguistics, 2018. p. 2–7. Disponível em: ⟨https://www.aclweb.org/anthology/P18-5002⟩.

GAŠIĆ, M. et al. Training and Evaluation of the HIS POMDP Dialogue System in Noise. In: *Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue*. Columbus, Ohio: Association for Computational Linguistics, 2008. p. 112–119.

GAŠIĆ, M. et al. Dialogue manager domain adaptation using Gaussian process reinforcement learning. *Computer Speech & Language*, v. 45, p. 552–569, 2017. ISSN 0885-2308.

GAŠIĆ, M.; YOUNG, S. Gaussian Processes for POMDP-Based Dialogue Manager Optimization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, v. 22, n. 1, p. 28–40, 2014.

GORDON-HALL, G.; GORINSKI, P. J.; COHEN, S. B. Learning Dialog Policies from Weak Demonstrations. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020. p. 1394–1405.

GORDON-HALL, G. et al. Show Us the Way: Learning to Manage Dialog from Demonstrations. *ArXiv:abs/2004.08114*, 2020.

GRIOL, D. et al. A statistical approach to spoken dialog systems design and evaluation. *Speech Communication*, v. 50, n. 8, p. 666–682, 2008. ISSN 0167-6393.

GU, J. et al. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016. p. 1631–1640. Disponível em: ⟨https://aclanthology.org/P16-1154⟩.

HAM, D. et al. End-to-End Neural Pipeline for Goal-Oriented Dialogue Systems using GPT-2. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020. p. 583–592. Disponível em: ⟨https://aclanthology.org/2020.acl-main.54⟩.

HAUSKNECHT, M. J.; STONE, P. Deep Recurrent Q-Learning for Partially Observable MDPs. In: *AAAI Fall Symposia*. [S.l.: s.n.], 2015.

HOSSEINI-ASL, E. et al. A Simple Language Model for Task-Oriented Dialogue. In: LAROCHELLE, H. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020. v. 33, p. 20179–20191. Disponível em: ⟨https://proceedings.neurips.cc/paper/2020/file/e946209592563be0f01c844ab2170f0c-Paper.pdf⟩.

JUMPER, J. et al. Highly acurate protein structure prediction with AlphaFold. *Nature*, v. 596, p. 593–589, 2021.

KAELBLING, L. P.; LITTMAN, M. L.; CASSANDRA, A. R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, v. 101, n. 1, p. 99–134, 1998. ISSN 0004-3702.

KIEFER, J.; WOLFOWITZ, J. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 23, n. 3, p. 462–466, 1952.

KINGMA, D. P.; BA, J. Adam: A Method for Stochastic Optimization. In: BENGIO, Y.; LECUN, Y. (Ed.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [S.l.: s.n.], 2015.

KOMATANI, K. et al. Multi-Domain Spoken Dialogue System with Extensibility and Robustness against Speech Recognition Errors. In: *Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue*. Sydney, Australia: Association for Computational Linguistics, 2006. p. 9–17.

Lecun, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

LEI, W. et al. 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)*. Melbourne, Australia: Association for Computational Linguistics, 2018. p. 1437–1447. Disponível em: ⟨https://aclanthology.org/P18-1133⟩.

LEVIN, E.; PIERACCINI, R.; ECKERT, W. Learning dialogue strategies within the Markov decision process framework. In: *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*. [S.l.: s.n.], 1997. p. 72–79.

LI, L.; WILLIAMS, J.; BALAKRISHNAN, S. Reinforcement Learning for Dialog Management using Least-Squares Policy Iteration and Fast Feature Selection. In: *10th Annual Conference of the International Speech Communication Association*. [S.l.: s.n.], 2009. p. 2475–2478.

LI, X.; CHEN, Y.-N.; LI, L. End-to-end task-completion neural dialogue system. In: *Proceedings of the The 8th International Joint Conference on Natural Language Processing*. [S.l.: s.n.], 2017. p. 733–743.

LIPTON, Z. et al. Bbq-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems. In: . [S.l.]: AAAI Press, 2018. p. 5237–5244.

LIU, B. et al. Dialogue Learning with Human Teaching and Feedback in End-to-End Trainable Task-Oriented Dialogue Systems. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018. p. 2060–2069.

MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 518, n. 7540, p. 529–533, fev. 2015. ISSN 00280836.

MO, K. et al. Personalizing a dialogue system with transfer reinforcement learning. In: *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*. [S.l.: s.n.], 2018.

NISHIMOTO, B. E. et al. Enhancing Designer Knowledge to Dialogue Management for Real-World Chatbots: A Comparison between Supervised and Reinforcement Learning Approaches. *Accepted at Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*, 2022.

NISHIMOTO, B. E.; REALI COSTA, A. H. Dialogue Management with Deep Reinforcement Learning: Balancing Exploration and Exploitation. In: *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.: s.n.], 2019. p. 449–454.

NISHIMOTO, B. E.; REALI COSTA, A. H. Deep Recurrent Q-Learning for Partially Observable Dialog System. In: *The Third Workshop on Reasoning and Learning for Human-Machine Dialogues (Deep-Dial2020) at the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*. [s.n.], 2020. Disponível em: ⟨https://sites.google.com/view/deep-dial2020/program⟩.

OPENAI. *OpenAI Five*. 2018. Disponível em: ⟨https://blog.openai.com/openai-five/⟩. Acesso em: 30-03-2019.

PENG, B. et al. Adversarial Advantage Actor-Critic Model for Task-Completion Dialogue Policy Learning. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2018. p. 6149–6153.

PENG, B. et al. Deep Dyna-Q: Integrating Planning for Task-Completion Dialogue Policy Learning. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, 2018. p. 2182–2192.

POWERS, D. M. W. The total turing test and the loebner prize. In: THE FLINDERS UNIVERSITY OF SOUTH AUSTRALIA. *98 Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*. [S.l.]: Association for Computational Linguistics, 1998. p. 279–280. ISBN 0-7258-0634-6.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning Representations by Back-Propagating Errors. *Nature*, v. 323, p. 533–536, 1986.

SAHA, T. et al. Towards integrated dialogue policy learning for multiple domains and intents using Hierarchical Deep Reinforcement Learning. *Expert Systems with Applications*, v. 162, p. 113650, 2020. ISSN 0957-4174.

SCHATZMANN, J.; YOUNG, S. The hidden agenda user simulation model. *IEEE Transactions on Audio, Speech, and Language Processing*, v. 17, n. 4, p. 733–747, May 2009. ISSN 1558-7916.

SCHULMAN, J. et al. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *ArXiv:abs/1506.02438*, 2015.

SCHULMAN, J. et al. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

SERBAN, I. V. et al. Building end-to-end dialogue systems using generative hierarchical neural network models. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2016. p. 3776–3783.

SHAWAR, B. A.; ATWELL, E. Different measurements metrics to evaluate a chatbot system. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. *Proceedings of the workshop on bridging the gap: Academic and industrial research in dialog technologies*. [S.l.], 2007. p. 89–96.

SILVER, D. et al. Mastering the game of go with deep neural networks and tree search. *Nature*, Nature Publishing Group, v. 529, p. 484–489, january 2016. Disponível em: ⟨https://doi.org/10.1038/nature16961⟩.

SINGH, S. et al. Reinforcement learning for spoken dialogue systems. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 1999. (NIPS'99), p. 956–962.

SU, P. et al. Continuously Learning Neural Dialogue Management. *CoRR*, abs/1606.02689, 2016.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. 1st. ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262193981.

TAKANOBU, R.; ZHU, H.; HUANG, M. Guided Dialog Policy Learning: Reward Estimation for Multi-Domain Task-Oriented Dialog. In: *EMNLP-IJCNLP*. [S.l.: s.n.], 2019. p. 100–110.

TAKANOBU, R. et al. Is your goal-oriented dialog model performing really well? empirical analysis of system-wise evaluation. In: *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. 1st virtual meeting: Association for Computational Linguistics, 2020. p. 297–310. Disponível em: ⟨https://www.aclweb.org/anthology/2020.sigdial-1.37⟩.

TIELEMAN, T.; HINTON, G. *Lecture 6.5-rmsprop: Divide the Gradient by Running Average of Its Recent Magnitude*. 2012. 26–31 p. Coursera course.

TOKIC, M. Adaptive ϵ-greedy Exploration in Reinforcement Learning Based on Value Differences. In: *Proc. of the 33rd Annual German Conference on Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer-Verlag, 2010. (KI'10), p. 203–210. ISBN 3-642-16110-3, 978-3-642-16110-0.

TURING, A. M. Computing machinery and intelligence. In: *Parsing the Turing Test*. [S.l.]: Springer, 2009. p. 23–65.

VASWANI, A. et al. Attention is all you need. In: GUYON, I. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. v. 30. Disponível em: ⟨https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf⟩.

VINYALS, O. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, v. 575, n. 7782, p. 350–354, 2019. ISSN 1476-4687. Disponível em: ⟨https://doi.org/10.1038/s41586-019-1724-z⟩.

VINYALS, O.; LE, Q. V. A neural conversational model. In: *ICML Deep Learning Workshop 2015*. [s.n.], 2015. Disponível em: ⟨https://arxiv.org/abs/1506.05869⟩.

VLASOV, V.; DRISSNER-SCHMID, A.; NICHOL, A. Few-Shot Generalization Across Dialogue Tasks. *CoRR*, abs/1811.11707, 2018.

VLASOV, V.; MOSIG, J. E. M.; NICHOL, A. Dialogue Transformers. *ArXiv*, abs/1910.00486, 2019.

WANG, S. et al. Task-completion dialogue policy learning via Monte Carlo tree search with dueling network. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, 2020. p. 3461–3471. Disponível em: ⟨https://aclanthology.org/2020.emnlp-main.278⟩.

WANG, W. et al. Incremental learning from scratch for task-oriented dialogue systems. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, 2019. p. 3710–3720. Disponível em: ⟨https://aclanthology.org/P19-1361⟩.

WANG, Z. et al. Policy Learning for Domain Selection in an Extensible Multi-domain Spoken Dialogue System. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: [s.n.], 2014. p. 57–67.

WEISZ, G. et al. Sample efficient deep reinforcement learning for dialogue systems with large action spaces. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, v. 26, n. 11, p. 2083–2097, Nov 2018. ISSN 2329-9290.

WEIZENBAUM, J. Eliza - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, New York, NY, USA, v. 9, n. 1, p. 36–45, 1966.

WEN, T.-H. et al. A network-based end-to-end trainable task-oriented dialogue system. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, 2017. p. 438–449. Disponível em: ⟨https://www.aclweb.org/anthology/E17-1042⟩.

WESTON, J.; CHOPRA, S.; BORDES, A. Memory Networks. In: BENGIO, Y.; LECUN, Y. (Ed.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [S.l.: s.n.], 2015.

WILLIAMS, J. D.; ASADI, K.; ZWEIG, G. Hybrid Code Networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, 2017. p. 665–677.

XU, Y. et al. Meta Dialogue Policy Learning. *ArXiv*, 2020.

YANG, S.; ZHANG, R.; ERFANI, S. GraphDialog: Integrating graph knowledge into end-to-end task-oriented dialogue systems. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, 2020. Disponível em: ⟨https://aclanthology.org/2020.emnlp-main.147⟩.

ZHANG, Z. et al. Recent Advances and Challenges in Task-oriented Dialog System. *Science China Technological Sciences*, 2020. ISSN 1674-7321.

ZHAO, T.; ESKENAZI, M. Towards End-to-End Learning for Dialog State Tracking and Management using Deep Reinforcement Learning. In: *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Los Angeles: Association for Computational Linguistics, 2016. p. 1–10.

ZHU KAIXIANG LIN, A. K. J. Z.; ZHOU, J. Transfer Learning in Deep Reinforcement Learning: A Survey. *ArXiv*, 2020.