Reinforcement Learning with Natural Language-based Exploration Strategies for Task-oriented Dialog Systems

Author:

Nicolas Benielli Borrajo

Supervised by:

Prof. Jun Wang (UCL)

Dr. Philip John Gorinski (Huawei Noah's Ark Lab)

Gabriel Gordon-Hall (Huawei Noah's Ark Lab)

MSc. Machine Learning

Department of Computer Science

University College London

August 2020

Abstract

Many end-to-end multi-task Dialogue Systems architectures rely on different modules to compose an entire dialogue pipeline. One of these components is a Policy module, which decides the actions to take given the inputs from the user. Moreover, the Policy can be learned by using Reinforcement Learning algorithms. These algorithms rely on an environment in which a learning Agent acts and receives feedback about the taken actions through a reward signal. However, most of the current Dialogue Systems environments only provide naïve (and sparse) rewards. The aim of this project is to explore Intrinsic Motivation Reinforcement Learning approaches where the agent learns an intrinsic reward signal, allowing it to predict the quality of its actions and speed up training. In particular, I extend Random Network Distillation and Curiosity-driven Reinforcement Learning algorithms to use (semantic) similarity between utterances to judge how often a state has been visited through a reward signal, which encourages exploration. Results obtained for MultiWOZ, a multi-domain dialogue data set, show that Intrinsic Motivation-based Dialogue Systems can outperform a policy that only observes extrinsic rewards. In particular, Random Network Distillation trained by taking the semantic similarity between user-system dialogues achieves an average success rate of 73 percent, improving significantly over the baseline PPO, which has an average success rate of 60 percent. Moreover, other performance metrics such as complete and book rates were also increased by 10 percent with respect to the baseline. I also show that these intrinsic motivation models make the system policy more robust to an increasing number of domains, suggesting that they may be a good approach for scaling up to environments containing an even bigger number of domains.

Acknowledgements

I would like to thank my industrial supervisors Philip John Gorinski and Gabriel Gordon-Hall from Huawei Noah's Ark Lab for their help and guidance. I also want to thank my classmates Simon, Petar, Bruno, Conrad and Nico for allowing me to share with them this amazing (and strange) past year. And I would like to especially thank Nhu, for all her support and love and for always believing in me.

Contents

List of Figures

List of Tables

Listings

Acronyms

DA: Dialogue Act

DM: Dialogue Manager

DS: Dialogue System

DST: Dialogue State Tracker

IC: Intrinsic Curiosity

ICM: Intrinsic Curiosity Module

LSTM: Long Short-Term Memory

MDP: Markov Decision Process

NLG: Natural Language Generation

NLU: Natural Language Understanding

POMDP: Partially-observable Markov Decision Process

PPO: Proximal Policy Optimization

ReLU: Rectified Linear Unit

RL: Reinforcement Learning

RND: Random Network Distillation

RNN: RecurrentNeural Networks

TRPO: Trust Region Policy Optimization

Utt: Utterances

Chapter 1

Motivation

In recent years, the field of human-computer interaction has been increasingly studied by Artificial Intelligence researchers. A problem that seems trivial at a first glance but in reality is far from simple is being able to develop *good* conversational programs, where a system converses with a user – a human – in order to achieve some goal. While there have been many different approaches to building conversational agents, Task-oriented Dialogue Systems rely on interacting with the user by asking questions and providing the requested information until the user's goal has been fulfilled. One of the key parts of a Dialogue System is the Dialogue Manager, which can take many forms from rule-based approaches (hand-crafted features) to (Deep) Reinforcement Learning policies that learn the best actions to take given the user's inputs. However, given the high complexity of Dialogue Systems used as environments for Reinforcement Learning, it remains an open problem how to find, train and fine-tune Reinforcement Learning algorithms that can learn optimal policies for maintaining good quality dialogues. In particular, many Dialogue System environments are currently modelled to provide a positive (extrinsic) reward only at the end of a successful dialogue, which leads to the well-known problem of reward sparsity in Reinforcement Learning. Moreover, if the environment presents a high action and state space dimensions and is multi-domain, it becomes nearly impossible for standard Reinforcement Learning algorithms – such as DQN – to learn at all.

The goal of this thesis is to explore different approaches to tackle the reward sparsity problem in Reinforcement Learning for Dialogue Systems, by integrating linguistic feedback into the training. While most of Reinforcement Learning environments are based on easily modelled phenomena, Dialogue Systems inherently rely on human language, which is notoriously hard to model and therefore to integrate into an environment. However, it is possible to make use of phenomena like semantic similarity to better handle the reward sparsity. I explore two main ideas that allow the system to generate internal (intrinsic) rewards for each dialogue turn: Random Network Distillation and Curiosity-based Reinforcement Learning. Both methods generate an auxiliary, step-wise reward that measures the novelty of the transition taken by the system. Furthermore, I implement these methods such that they can learn directly from the agent-environment semantic utterances, as well as from their dialogue acts. I show that by doing so the system increases its learning capabilities and can achieve better results than simple baseline models such as Proximal Policy Optimisation, which is ultimately the result of improving the policy exploration in the semantic space during training.

In Chapter 2 I introduce the background concepts that this thesis is built upon, with a description of the components that make a Dialogue System, a review of Reinforcement Learning, and how the policy in a Dialogue System can be learned by using Deep Reinforcement Learning. In Chapter 3 I define the methodologies I follow in the experiments, as well as the task-oriented Dialogue Systems library that is used. In particular, I describe in detail the Proximal Policy Optimisation algorithm, that is used as the Reinforcement Learning baseline to learn the system policy. Also, I introduce and detail Random Network Distillation and Intrinsic Curiosity. In Chapter 4 I describe the experimental setup, and provide all the hyperparameters and overall training and evaluation procedure used in this work. Results are shown in Chapter 5, where not only the learning curves and model performances are shown, but also sensitivity analysis of the learned policies and some concrete dialogue examples, to showcase how different policies behave under similar con-

ditions. Finally, in Chapter 6 I summarise the findings of this project, discuss the limitations of the proposed methods encountered, and are the potential improvements to follow up my work. Corresponding code for this project can be found in `https://github.com/thenickben/ucl_thesis/`.

Chapter 2

Background

2.1 Task-oriented Dialogue Systems

- **User**: Alright book a taxi for me as well . I 'll need to get to the two places and I'm not familiar with the town . Thanks . The taxi should go to pizza express Fen Ditton . I need a taxi to get me from india house. I want to leave by 07:45 .
- **System**: When would you like to arrive ? Where will you leave from ? Where will the taxi be taking you ? What time do you want to leave by ?.
- **User**: Still need a cab by my booked time , 07:45 please. I need a tax to pizza express Fen Ditton . I'm at india house. I don't care about the arrive of the taxi.
- **System**: volvo is booked for you . The contact is 15372986675 . You are welcome . Is there anything else I can help you with today ?
- **User**: Thank you , goodbye.
- **System**: Thank you for using our system !

Figure 2.1: Example of a user-system dialogue

Task-oriented Dialogue Systems (DS)[1] are computer programs that help a user, typically a human, to achieve their specific goals while interacting purely in the form of natural language (i.e. in plain English). Opposite to task-oriented DS, there exist another popular type of DS known as *conversational AI*, *chit-chat dialog systems*, or simply *chatbots*. In these systems, the goal is to extend the dialogue as much as possible without any kind of goal centered around helping the user. This thesis is focused on task-oriented DS, and a simple example of what a dialogue between

[1] I will sometimes refer to the DS simply as the *system*.

a user and the system could look like[2] is shown in Figure 2.1. The first thing to notice is that the system must have the ability of interpreting what the user's goals or **intents** are and to which **domains** they belong to. As can be seen this above example, the user gives all the necessary information to the system in their first utterance (also known as a *dialogue turn*). At this point, it is unknown to the system what the goals of the user are. However, given that it identified the taxi domain, the first output from the system is to ask some basic follow-up questions: *where from, where to, when is the pick-up, when is the arrival*. Even if most of these inputs were already given by the user (e.g. `"india house"`), the system seems to ask some pertinent questions again (red) and includes one that may have been missed by the user (green), like what the desired arrival time is, whereas it does not ask for a destination. In fact, the steps that the system follows during the dialogue are to identify the domain, classify the intent, and ask for and retrieve the missing information through interaction with the user. This procedure in DS is known as *slot filling*.

After the first output from the system, the user provides (in some cases, again) the requested information, letting the system know that the arrival could be at any time[3]. With all this collected information, the system proceeds to book the taxi and informs the user about the car type and the phone number, and it asks the user if further help is needed. At this point, the user could proceed with another request, for example booking another taxi from the restaurant to a cinema, booking cinema tickets, and one more taxi from the cinema to their hotel. If this was the case, the system should follow up the conversation by identifying the new domains, intents and interacting with the user while filling the remaining slots. However, the user thanks the system for its help, concluding the dialogue instance.

In general, user-system interactions in Task-oriented DS are structured in the following way:

[2]This example was taken at random from a model trained for this thesis.

[3]While this is a fictitious dialogue based on the limited dataset used in this thesis, real-world DS could need to know the desired arrival time in order to request a quote from different taxi agencies, and pick the most convenient one. Moreover, solely based on the information provided by the user, the system could book a table at this restaurant at, for example, three hours after the pick-up time, allowing the taxi agency to profit from a very slow and painful commute for the user.

- **Domains**: the different dialogue domains that are supported by the DS (e.g. `"cinema"`, `"airplane"`, `"hotel"`, etc).

- **Intents**: within a particular domain there are different types of intents. For example, for the domain `"hotel"` a user may want to book a room (intent:`"book"`) or simply request a price range for hotels in a certain area (intent:`"request"`).

- **Slots**: for every intent within a particular domain there is specific domain-intent information. For example, in (intent:`"request":"phone number"`), the slot value for the `"request"` intent would be `"phone number"`.

- **Dialogue Action**: also known as *dialogue acts* (or DA for short), represent any admissible[4] triple (*intent, slot, value*).

A practical way to represent a particular goal for the user at the beginning of the dialogue is by defining a *dictionary* with N possible domains, where each domain can have different intents and corresponding slots, as shown in Listing 2.1 below. Going back to the taxi booking dialogue example, the initial goal of the user (unknown to the system) can be represented as shown in Listing 2.2, where the `'?'` indicate those slots that have to be filled in by the system and responded to the user. During the progress of the dialogue, the DS will increasingly fill the slots until no remaining information is required from the user, hence terminating the dialogue as a successful one. For the case of the taxi booking example, the user has all the requested information at the end of the dialogue as shown in Listing 2.3.

In practice, many DS are trained using datasets that usually consist of human to human dialogues. Thus, each dataset will have their own domains and related intents with its possible slots, which is known as the **Domain Ontology** of the dataset.

[4]It should be noticed that not any triple (intent, slot, value) can form a valid DA for a given dialogue dataset. For example, ('book', 'police', 'Indian food') would not be admissible by any well-defined Domain Ontology.

```
{   'domain 1': {'intent 1': {'slot 1': 'value',
                              'slot 2': 'value',

                              'slot h': 'value'
                             },


                 'intent m': {'slot 1': 'value',
                              'slot 2': 'value',

                              'slot j': 'value'
                             }
                },

     'domain N': {'intent 1': {'slot 1': 'value',
                              'slot 2': 'value',

                              'slot k': 'value'
                             }
                }
}
```

Listing 2.1: Generic example of a Domain Ontology instance structure

```
{'taxi': {'info': {'departure': 'india house',
                   'destination': 'pizza express Fen Ditton',
                   'leaveAt': '07:45'},
          'reqt': {'car type': '?', 'phone': '?'}}}
```

Listing 2.2: Goal for user in taxi booking example at the beginning of the dialogue

```
{'taxi': {'info': {'departure': 'india house',
                   'destination': 'pizza express Fen Ditton',
                   'leaveAt': '07:45'},
          'reqt': {'car type': 'volvo', 'phone': '15372986675'}}}
```

Listing 2.3: Goal for user in taxi booking example at the end of the dialogue

2.1.1 Dialogue Systems architecture

The general architecture of a DS consists of different interacting components (known as *pipeline approach*), where the end-to-end functionality is to receive the natural language input utterances from the user, interpreting what the user wants while taking the necessary actions, and responding to the user by using natural language sentences as well[5]. These components are:

[5]Although the user inputs can be spoken utterances, in this thesis I will only consider the case where these utterances occur in written text. Any form of speech recognition, synthesis and generation will be ignored here, as it constitutes a very different (although interesting!) problem

- **NLU**: The Natural Language Understanding (NLU) component, which is used to parse the intent of the other agent[6], takes an utterance as input and outputs the corresponding dialogue acts.

- **DST**: The Dialogue State Tracking (DST) component updates the *belief state*, which contains the user goals and the dialogue acts from both agents during the dialogue.

- **Policy**: It receives the belief state during each dialogue turn, and decides what the best outputs are. This thesis focuses on modelling this component in particular.

- **NLG**: The Natural Language Generation (NLG) component transforms dialogue acts into Natural Language sentences (utterances).

On top of these components, a **user simulator** is commonly implemented to simulate a user in order to interact with the system. By doing so, user goals can be sampled to generate natural language sentences (utterances).

While *monolithic* DS ([62], [28], [18], [34], [27]) do not rely on any pipeline-based architecture, they can lack of explainability and may be treated as a black box. Segmenting a DS into individual components allows for a better interpretation, tuning and control of the DS as a whole. Moreover, it makes it possible to learn the Policy using Reinforcement Learning methods, which will be discussed in Section 2.2.

It is common to refer as the Dialogue Manager (DM) to both DST and Policy together. The interaction among the components of a DS is shown in Figure 2.2. In the remainder of this section, I will give short descriptions of these components and some of the approaches explored in academia for each one of them. For an up-to-date survey of recent advances and challenges in task-oriented DS, see [74].

[6]I will refer indistinctly as agent to either the user or the system, unless it is necessary to specify which agent in particular I am referring to. Therefore, a dialogue happens between two interacting agents.

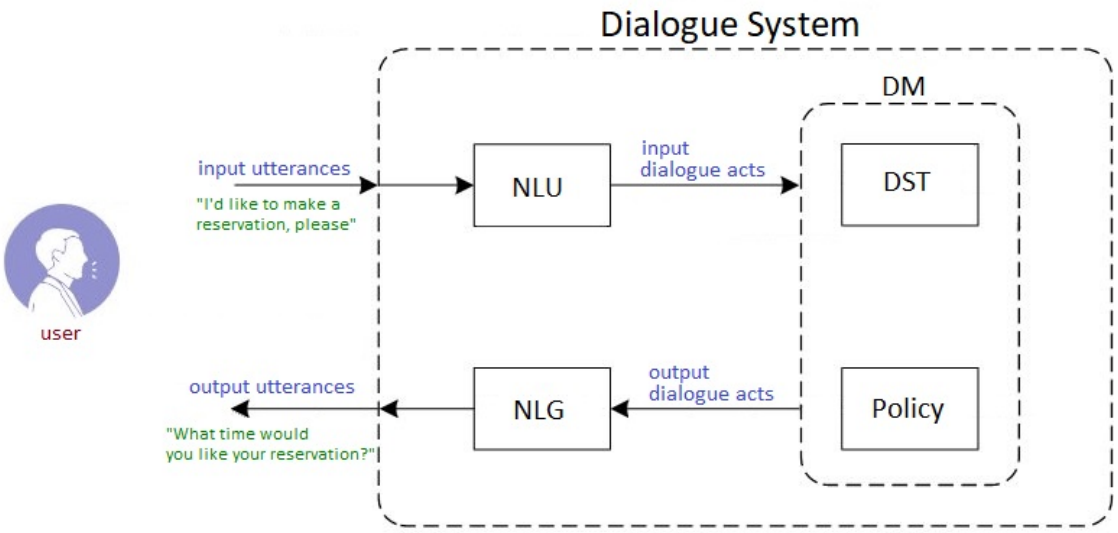

Figure 2.2: Dialogue System pipeline

2.1.2 User Simulator

Learning from real users is a costly and time-consuming task. Moreover, if the Policy module is trained by using Reinforcement Learning (RL, see Section 2.2), many interactions with the environment (which in the case of DS it would be the user) are needed, making it impossible to train by interacting with real users. Instead, the DS interacts with a simulation of a real user, which is known as the *User Simulator*. Real world dialogues are collected into a database (known as *Dialogue Corpus*) that then is used to construct a model that simulates goals and utterances from real users (see Figure 2.3).

There are two main approaches when building a User Simulator: *agenda-based* and *model-based* methods. The agenda-based approach ([65]) assumes that the user state can be represented by a *Goal* (for example the one shown in Listing 2.2) and an *Agenda*, which is a dictionary-like structure that updates the user goals during the course of the dialogue while interacting with the system. In [48] the authors formalises the dialogue on a semantic-level Partially-observable MDP (POMDP) and propose a variation of the EM algorithm to learn a probability distribution over dialogue states. However, as discussed in [22], this approach is limited in terms of the complexity of interactions between users that, for example, could change their

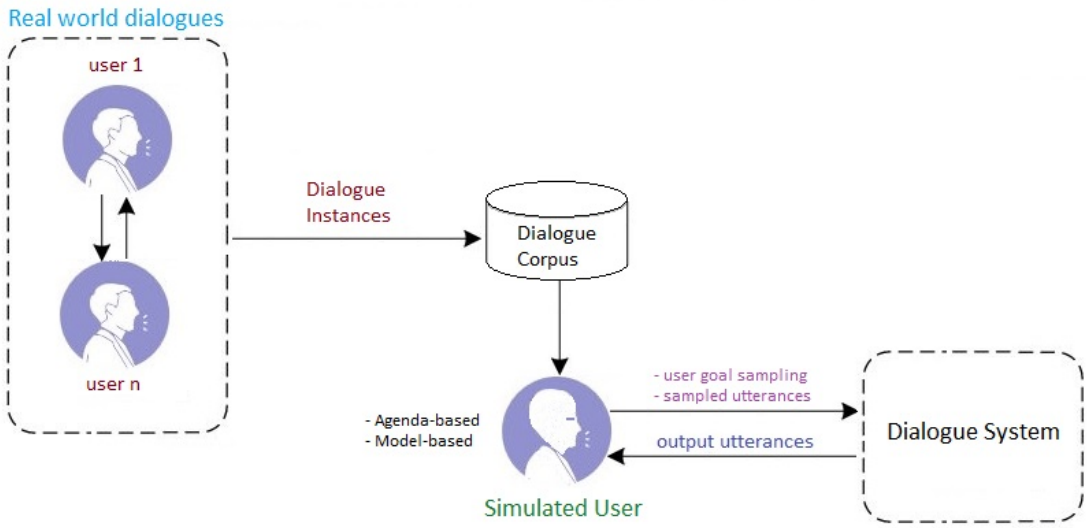

Figure 2.3: User Simulator

goals during the course of the dialogue. Therefore, in [22] the authors incorporate random decision points during the dialogue, parametrised by a set of parameters that are estimated from real user data using a sample-based maximum likelihood approach. In general, while this agenda-based approach is simple to implement and produces coherent goals, it suffers from the need to design hand-crafted rules which are domain dependent, thus limiting its scalability, flexibility and generalisation to multiple domains.

Model-based user simulators are built in a data-driven way, without explicitly relying on hand-crafted rules. In [1] a sequence-to-sequence model is used to train a user simulation in the restaurant search domain, taking the dialogue context into consideration without the help of external data structure. [24] introduced the Neural User Simulator (NUS) which learns user behaviour from a corpus and generates natural language utterances directly instead of dialog acts. [11] uses imitation learning to train a user simulator, and while it shows good results for a simple tourist information domain, training utility function-based models as in the case of Inverse RL requires a huge amount of computing time. In [54] the authors use Sequicity ([26]) to construct a model-based user simulator which is trained using supervised learning.

2.1.3 Natural Language Understanding

The Natural Language Understanding (NLU) component takes user utterances as input and transforms them into their corresponding dialogue acts. In order to achieve this transformation, the NLU module has to classify the domains and intents, and extract the slots values to be filled during the conversation. Thus, any NLU module is limited to a domain ontology. [44] uses (vanilla) Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) models for intent classification, while [35] uses RNN for slot filling tasks. [32] introduces a model which is a Semantic Tuple Classifier, discriminative semantic classification models that are used recursively to parse the semantic trees from input sentences. [77] introduces BERTNLU, which is based on a pre-trained BERT model ([12]) with the addition of extra classifier layers for intent classification and slot tagging, which is fine-tuned on a specific dialogue corpus, achieving good results in comparison to other models.

2.1.4 Dialogue Manager

The Dialogue Manager (DM) takes the dialogue acts generated by the NLU module and it determines the next action to take. This is, what is the best dialogue act to output given the context and the history of each dialogue instance. It is composed of two modules: the Dialogue State Tracker (DST) and the Policy.

2.1.4.1 Dialogue State Tracker

The DST extracts all the relevant information from the dialogue acts it receives, such as intents, slots and values. It creates a representation of the user's goal (known as the *belief state*) and keeps track of the history of the dialogue. The simplest approach of DST models ([37], [20], [75]) assumes that the dialogue ontology is given in advance, where all possible combinations of intents, slots and values are known. However, real world dialogues are usually *multi-domain*, where different domains, intents and slot/value pairs can be contained in a single utterance (for example MultiWOZ dataset [8], introduced in Section 3.1). Moreover, it is not always possible or realistic to assume full knowledge about the domain ontology ([73]). Therefore, [72] introduces TRADE, a DST model where different domains

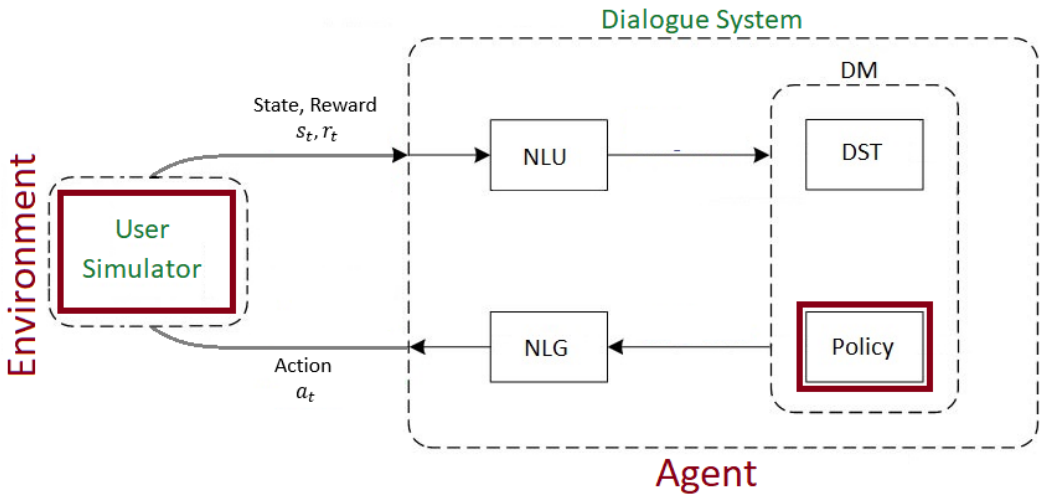

Figure 2.4: Dialogue System seen as a Reinforcement Learning problem

share parameters (so that it is more robust under unseen values) which does not rely on a pre-defined ontology. While TRADE is considered the current state-of-the-art model in multi-domain DST, achieving a joint accuracy of 45.60% on the MultiWoz dataset, in [19] the authors propose a model, named *TripPy*, based on a set of *copy mechanisms strategies* that allow it to handle different situations when the slots are filled [7]. TripPy achieves a joint accuracy of 55.30% on the MultiWoz dataset.

2.1.4.2 Policy

The Policy component of a DS is in charge of deciding what information the system should retrieve from or provide to the user. It receives the belief state from the DST and outputs dialogue acts. In recent years, Reinforcement Learning has become a popular way to train the Policy component of a DS. Figure 2.4 shows a representation of the DS architecture as an RL problem, where the User Simulator is the environment, the DM is the agent, and the Policy inside the DM can be learned by RL.

Among the first approaches using RL in the DS scope are [55] and [42], where task-oriented DS are set in an MDP framework and solving it using standard Q-

[7] citing from the paper: *"The intuition is that values are either explicitly expressed by the user, that they are expressed by the system and referred to by the user via confirmation or rejection, or that they have been expressed earlier in the dialog as assignment to another domain-slot pair (coreference)."*

learning ([64]) algorithms, while in [29] the RL problem is solved by using a least-squares optimisation approach. [30] introduce a Thompson sampling-based exploration methodology, showing that it allows for faster learning than in traditional exploration methods such as ε-greedy. Additionally, as their base RL algorithm is DQN ([36]) and therefore uses a replay buffer, they show that pre-filling it with a few samples of successful dialogues can improve the learning speed.

Pre-training the policy before starting the learning by RL is crucial for ensuring that the policy learns at all. One approach is to use imitation learning to replicate a given expert policy. [17] takes the idea of Deep Q-Learning from Demonstrations ([21]), using rule-based expert demonstrations to guide the policy learning process, whereas in [16] the authors move away from rule-based experts to weaker demonstrations and also propose a fine-tuning algorithm inspired by [45], which allows to improve the demonstrations gathered by those weaker experts.

A different approach to speed up the policy training is to consider different reward functions. [56] uses two neural networks to assess the success of a dialogue following a supervised learning approach that relies on labelled data, while in [61] a different reward estimation based on the *interaction quality* between user and system is introduced. In [69] the ACER ([63]) RL algorithm is used in conjunction with Intrinsic Curiosity ([41]) for a simple domain (*Cambdridge Restaurants*, which is now part of MultiWOZ) to generate intrinsic rewards in order to reduce the sparsity in traditional DS, where there is usually a positive reward only at the end of a successful dialogue.

A more detailed and up to date review of other methods for policy learning can be found in [14].

2.1.5 Natural Language Generation

The Natural Language Generation (NLG) component works in the opposite way of NLU, taking dialogue acts as inputs and transforming them into natural language utterances. The simplest approach is to use a *Template-based* model, where the dialog acts are passed to a template with pre-defined utterances where the relevant slots (given each particular dialogue act) are filled to compose sentences. For example,

let's assume a dialogue act that the system outputs is

```
{inform:"numberofrestaurants" 18;
 inform:"pricerange", "moderate";
 request:"numberofpeople", "?"
}
```

Listing 2.4: Example of a dialogue act to be used for utterances generation

then all the following are valid utterances under this dialogue act as constraint:

- " I've found 18 restaurants that satisfy your criteria, but they are in the moderate price range. What is the price range you are looking for?"
- " There are 18 restaurants in the moderate price range. For how many people do you want the booking? "
- " There are 18 restaurants for your price range, but I can suggest La Parraca, which is a small barbeque restaurant located in Belgrano, and is in the moderate price range. How many people should I book for? "

By using a pre-defined template with natural language structures where the relevant information can be filled in to form complete utterances, the process of NLG is straightforward and fast to compute. However, this approach suffers from the limitation of being domain-dependant and not easily scalable. A different approach, known as *corpus-based* method, is to learn how to generate utterances directly from the data, as in [66] and [33], where ranking methods are used to select the best sentences that are stochastically generated by a language model, while in [67] the authors extend this methodology by using conditioned language models. For a more detailed review of the different existing NLG methods, see [15].

2.1.6 Evaluation metrics for Dialogue Systems

Evaluating the performance of a DS is a challenging task, given the many sources of errors that can be present in it. While different components such as NLU and NLG can be evaluated independently of the results of the DS, direct automatic metrics are frequently used to evaluate the DS performance as a whole.

In particular, BLEU ([40]) computes the word overlap between the generated output and the reference response, while Inform rate and Success rate ([8]) are also used. Inform rate measures how often the system gives the user the requested information, whereas Success rate indicates how often the system correctly answers all the requested attributes. [34] uses a combined score defined as $\texttt{BLEU} + 0.5(\texttt{Success rate} + \texttt{Inform rate})$ as the reported evaluation metric. However, as discussed in [49], policies learned with a poor user simulator model may perform well when tested using this same simulator, but perform poorly when they are tested using better ones. A survey on metrics for the evaluation of user simulations can be found in [43] .

2.2 Reinforcement Learning

As mentioned in the previous section, the Policy component of a DS is in charge of deciding what information the system should provide or request at any dialogue turn. It can be seen as a function that takes observations from the user and outputs an action (e.g. answering a question or providing requested information). Thus, the Policy module within a DS has to *learn* what the best actions are to take, given the different user requests. One way of doing so is to consider this as a Reinforcement Learning (RL) problem. In this section I review general RL definitions and methods that will be used later in this work, while keeping an abstraction from DS as a particular application or RL.

2.2.1 Key Concepts and Terminology

In its basic formulation, a Reinforcement Learning problem is composed of an *agent* interacting with an unknown *environment*. The agent takes *actions* based on its *observations* from the environment, and the environment will consequently act by changing its *state* to a new one and providing back a *reward* to the agent. The ultimate goal for the agent is to maximise the rewards collected by interacting with the environment. This continuous interaction can be represented as in Figure 2.5.

It should be noted, however, that states are intrinsic to the environment, while the agent only receives observations from it. Depending on the ability of the agent of

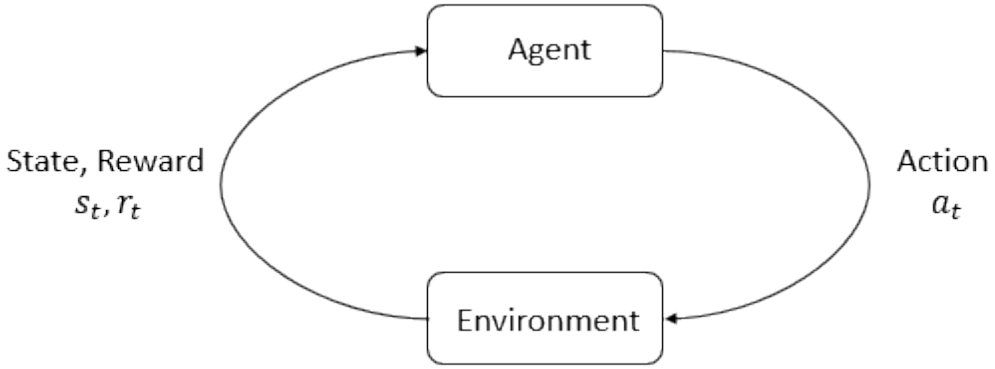

Figure 2.5: Reinforcement Learning setup

observing the environment, a distinction is made between *fully-observable environments*, where the observations of the agent are exactly the environment states, and *partially-observable environments*, where the agent can only observe a partial representation of the true environment states. In the remainder of this section I will refer to states and observations indistinctly.

Definition 2.2.1. A discrete-time Markov Decision Process (MDP) is a tuple $(\mathbb{S}, \mathbb{A}, \mathbb{R}, \mathbb{P}, \gamma)$, where:

- $\mathbb{S}$ is the set of all possible states the environment can be in;
- $\mathbb{A}$ is the set of all possible actions the agent can take;
- $\mathbb{R} : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathbb{R}$ is the reward function;
- $\mathbb{P}(r, s' \mid s, a)$ is the joint probability of a reward $r \in \mathbb{R}$ and next state $s' \in \mathbb{S}$, given a state $s \in \mathbb{S}$ and action $a \in \mathbb{A}$,
- $\gamma \in [0, 1]$ is a discount factor that determines the priority of short-term rewards.

Definition 2.2.2. A *transition* is a tuple[8] (s, a, s', r). The agent observes the current state s, takes an action a, and consequently receives an immediate reward r. On the other hand, after receiving the action taking by the agent, the environment will change its state into s'.

[8]While the discount factor γ can also be time-dependent, it is a deterministic function for the agent and therefore I do not include it into the transition tuple.

In particular, an MDP satisfies the Markov Property, where the conditional probability distribution of future states and rewards depends only on the present state and not on past transitions.

Definition 2.2.3. A *trajectory* τ is defined as the concatenation of consecutive transitions within an MDP, $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_t, a_t, r_t)$.

Definition 2.2.4. A *terminal state* is a state that, once reached, terminates the interaction between the agent and the environment. The time step t where a terminal state is reached is denoted by $t = T$.

Definition 2.2.5. An *episode* is a trajectory that reaches a terminal state, e.g. $(s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T, a_T, r_T)$

Definition 2.2.6. A *policy* is a function $\pi : \mathbb{S} \to \mathbb{A}$ that maps states into actions.

Definition 2.2.7. A policy can be either *deterministic* $\pi(s) = a$, where for any state observed by the agent it will always take the same action, or *stochastic* $\pi(a \mid s) = \mathbb{P}_\pi[A = a \mid S = s]$, where the actions taken follow a certain probability distribution $\mathbb{P}_\pi$.

Definition 2.2.8. The *return* $G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$ is the cumulative discounted reward of a given trajectory τ. Whether the trajectory has a terminal state or not, the return will be finite given that the discount rate $\gamma \in [0, 1]$.

Definition 2.2.9. An *optimal policy* π^* is such that, for any admissible trajectory within the MDP, it maximizes the expected return:

$$\pi^* = \arg\max_\pi \mathop{\mathbb{E}}_{\tau\sim\pi}[G(\tau)] = \arg\max_\pi \mathop{\mathbb{E}}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r_t] \tag{2.1}$$

Definition 2.2.10. The *state-value function* (also simply known as the value function) $v(s)$ is defined as the expected return starting from a particular state s_0 and following a policy π:

$$v_\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[G(\tau) \mid s_0 = s] \tag{2.2}$$

Definition 2.2.11. The *action-value function*[9] $q(s,a)$ is defined as the expected return starting from a particular state s_0 and taking an action a_0 while following a policy π:

$$q_\pi(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}[G(\tau) \mid s_0 = s, a_0 = a] \tag{2.3}$$

Definition 2.2.12. The difference between $q_\pi(s,a)$ and $v_\pi(s)$ is known as the *advantage function* $A_\pi(s,a)$, which measures how good taking a particular action is with respect to the others, on average:

$$A_\pi(s,a) = q_\pi(s,a) - v_\pi(s) \tag{2.4}$$

It should be noted that the state-value function $v_\pi(s)$ can also be defined as the expectation of the action-value function $q_\pi(s,a)$, this is

$$v_\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi} q(s,a) \tag{2.5}$$

By unrolling the return G_t at some point in time t into the sum of the discounted rewards, the value functions $q(s,a)$ and $v(s)$[10] can be expressed recursively:

$$\begin{aligned} v(s) &= \mathbb{E}[G_t \mid s_t = s] \\ &= \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_t = s\right] \\ &= \mathbb{E}[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \ldots) \mid s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma G_{t+1} \mid s_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma v(s_{t+1}) \mid s_t = s] \end{aligned}$$

and

$$\begin{aligned} q(s,a) &= \mathbb{E}[r_{t+1} + \gamma v(s_{t+1}) \mid s_t = s, a_t = a] \\ &= \mathbb{E}\left[r_{t+1} + \gamma \mathop{\mathbb{E}}_{a\sim\pi} q(s_{t+1}, a) \mid s_t = s, a_t = a\right] \end{aligned}$$

[9] It is common in RL referring to both $v(s)$ and $q(s,a)$ simply as the *values*

[10] Unless it is necessary, I will omit the policy π subscript in order to simplify the notation.

Definition 2.2.13. The recursive expressions for the value functions are known as the *Bellman Expectation equations*:

$$v(s) = \mathbb{E}\left[r_{t+1} + \gamma v\left(s_{t+1}\right) \mid s_t = s\right] \tag{2.6}$$

$$q(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \mathop{\mathbb{E}}_{a\sim\pi} q\left(s_{t+1},a\right) \mid s_t = s, a_t = a\right] \tag{2.7}$$

When the policy is optimal, $\pi = \pi^*$, the value functions are maximised, representing the best return it can be obtained at a given state and taking a particular action while following an optimal policy

$$v^*(s) = \max_{\pi} v_\pi(s) \tag{2.8}$$

$$q^*(s,a) = \max_{\pi} q_\pi(s,a) \tag{2.9}$$

In fact, the optimal policy can be found by maximising either value function:

$$\pi^* = \arg\max_{\pi} v_\pi(s) \tag{2.10}$$

$$\pi^* = \arg\max_{\pi} q_\pi(s,a) \tag{2.11}$$

Definition 2.2.14. The recursive expressions for the optimal value functions are known as the *Bellman Optimallity equations*:

$$v^*(s) = \max_{a} \mathbb{E}\left[r_{t+1} + \gamma v^*\left(s_{t+1}\right) \mid s_t = s\right] \tag{2.12}$$

$$q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a} q^*\left(s_{t+1},a\right) \mid s_t = s, a_t = a\right] \tag{2.13}$$

Solving an MDP means finding an optimal policy, such that the agent will always take optimal actions a^* for any observation from the environment

$$a^* = \pi^*(s) \tag{2.14}$$

Note that in general a policy π is better than π', $\pi \geq \pi'$, if:

$$v_\pi(s) \geq v_{\pi'}(s), \forall s \tag{2.15}$$

Therefore, they can be many optimal policies, but all of them will have the same optimal values. If the model of the environment is unknown to the agent, the problem is known as *model-free*. There are two main ways of solving it:

- Value function optimization, where the Bellman equations are used to either find $v^*(s)$ or $q^*(s,a)$;
- Policy optimization, where the policy is parametrised and directly optimised without relying on any value function.

These approaches rely on storing the values in a table, which is efficient enough for small and discrete state-action spaces. Also, both approaches can be combined. See [58] for an in-depth review of RL algorithms.

2.3 Deep Reinforcement Learning

When the state-action space is large enough, storing the values in a table is not efficient. Instead, a more efficient approach is to approximate them using parametrised functions. This is known as *value function approximation*.

2.3.1 DQN

DQN ([36]) was the first algorithm to use a (deep) neural network to parametrise the Q-value function $Q(s,a,\theta)$, where its update follows a *Q-learning* algorithm ([64]):

$$q(s_t,a_t) \leftarrow q(s_t,a_t) + \alpha\left(r_{t+1} + \gamma \max_{a\in\mathbb{A}} q(s_{t+1},a) - q(s_t,a_t)\right) \tag{2.16}$$

where α is known as the *learning rate*. The authors introduced the following concepts to stabilise the training procedure:

- *Experience Replay*: Environment transitions (s_t,a_t,s_{t+1},r_t) are stored in a

replay memory from where random tuples are drawn during learning. This allows to de-correlate consecutive transitions and improves data efficiency.

- *Target network*: Instead of using one neural network $Q(s,a,\theta)$, there are two different networks using a different set of parameters θ, and the loss function for the Q-learning update becomes

$$\mathscr{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r+\gamma\max_{a'} Q\left(s',a';\theta^{-}\right) - Q(s,a;\theta)\right)^2\right] \quad (2.17)$$

  where θ' are the parameters of the target network, that is periodically updated to θ, and $U(D)$ is a uniform distribution over the transitions in the replay memory. Periodical updates of the target network allow to improve stability from short-term oscillations.

The loss function can be optimised using any gradient descent algorithm (see [46] for a review of the most frequently used ones). Exploration is achieved by using an ε-*greedy* algorithm to select actions, where with a (small) probability ε a random action is selected, otherwise selecting the greedy action $a = \arg\max_a Q(s,a,\theta)$ with a probability $1-\varepsilon$.

One of the main limitations of DQN is being designed only for discrete action spaces. When the actions to be taken are continuous (for example, rotation angles in a robotic arm) it is not possible to use the ε-greedy algorithm for exploration. In this case, different approaches are needed to find the optimal policy, being Policy Gradient method a commonly used one.

2.3.2 Policy Gradient

Instead of parametrising the value functions, Policy Gradient methods parametrise and optimise the policy directly. By doing so it makes possible to consider both discrete and continuous action and state spaces. This approach turns the reward

function into a parametrised $J(\theta)$[11]:

$$J(\theta) = \sum_{s \in \mathbb{S}} d^{\pi}(s) \sum_{a \in \mathbb{A}} \pi_{\theta}(a \mid s) Q^{\pi}(s,a) \tag{2.18}$$

where $d^{\pi}(s)$ is the stationary distribution for π_{θ}.

Definition 2.3.1. The *policy gradient theorem* allows to analytically compute the gradient of the reward function, as:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} \left[Q^{\pi}(s,a) \nabla \theta \ln \pi_{\theta}(a \mid s) \right] \tag{2.19}$$

See [57] for a proof of this theorem. It should be noted that optimising the reward function can be solved by gradient ascent methods, as in this case the goal is to maximise rewards.

2.3.3 REINFORCE

A direct application of the policy gradient theorem is to approximate $Q^{\pi}(s,a)$ by sampling trajectories, using Monte-Carlo methods, to compute their total return G_t[12]. Therefore, the gradient of the reward function becomes:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} \left[G_t \nabla \theta \ln \pi_{\theta}(a \mid s) \right] \tag{2.20}$$

This is known as the *REINFORCE* algorithm ([70]), and, while simple, it suffers from high variance in the returns induced by Monte-Carlo sampling methods. One way of reducing the variance is to introduce a *baseline* b_t, where instead the rewards r_t are taken as $r_t - b_t$. This baseline introduction keeps the policy gradient unbiased, and both q-value and advantage functions are common choices for the baseline.

2.3.4 Actor Critic

Another approach, that uses direct policy optimisation assisted by a value estimation algorithm, is known as *Actor Critic* (AC, [23]). It is composed by two modules:

[11] In the case of continuous action or state spaces, summations can be expressed by integrals, given that the policy $\pi_{\theta}(a \mid s)$ is a continuous function.

[12] Given that $Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t \mid s_t, a_t]$

- *Actor*: a parametrised policy $\pi_\theta(a \mid s)$, that is learned using a loss based on the policy gradient theorem (Equation 2.19) by gradient ascent;

- *Critic*: a parametrised value function, $Q_\Phi(s,a)$ or $V_\Phi(s)$, that is learned using a similar loss function as in Equation 2.17 by gradient descent.

Combining direct policy optimisation with the help of a Critic network that produces more stable estimates for the value function makes Actor Critic algorithms achieve lower variances than their *vanilla* version (such as REINFORCE).

Chapter 3

Methodology

3.1 MultiWOZ dataset

The MultiWOZ ([9]) dataset consists of human to human conversations happening at an information centre, which spans into seven domains: attraction, hospital, police, hotel, restaurant, taxi, train. It is a standard data set used in task-oriented dialogue systems research, and brings a realistic framework with over eight thousand dialogue instances which can be either single or multi-domain. Figure 3.1 shows the distribution of domains within the dataset. It can be seen that the majority of dialogue instances cover attraction, hotel, train and restaurant domains, while police and hospital domains occur less frequently. Figure 3.2 shows the distribution of the number of domains contained in each dialogue instance in the dataset. Nearly half of the dialogues span on two domains, while single domain dialogues represent nearly thirty percent of the dataset followed by dialogues with three domains as the least frequent case. There are no dialogues with more than three domains [1]. The dialogues are annotated as an initial goal followed by user and system dialogue acts.

[1] Shown distributions are the result of 2000 initial goals sampled from the MultiWOZ dataset version used in Convlab-2 (`https://github.com/thu-coai/ConvLab-2`), while in the original MultiWOZ dataset (`https://github.com/budzianowski/multiwoz`) the authors state that dialogues span between two and five domains.

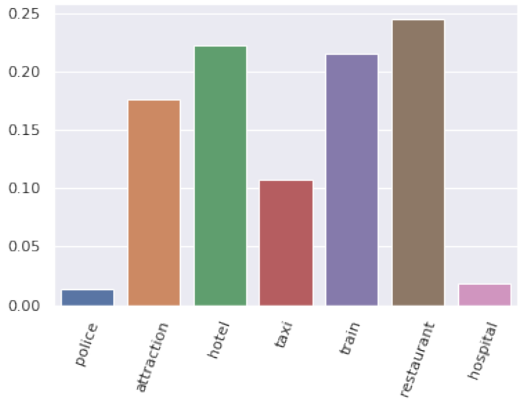

Figure 3.1: Distribution of domains in MultiWOZ dataset

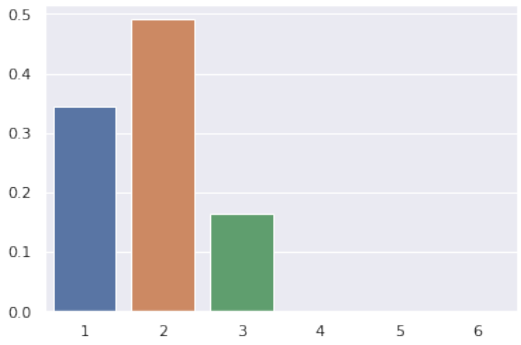

Figure 3.2: Distribution of number of domains per dialogue instance in MultiWOZ dataset

3.2 Convlab-2: Open Source Library for Dialogue System Research

Given the increasing attention that Task-oriented DS have gained during the last years, and due to their high complexity and many different models that can be adopted for each DS pipeline component, efforts were made in [78] to produce Convlab-2, an open-source, state-of-the-art DS library that allow researchers to quickly build, train and evaluate DS on multiple datasets. Convlab-2 was developed as the common framework for the Ninth Dialog System Technology Challenge (DSTC9) and is the successor of Convlab ([25]), introduced in the DSTC8. It supports four large-scale dialogue datasets: CamRest676 ([68]), MultiWOZ ([9]), DealOrNoDeal ([27]), and CrossWOZ ([76]), and allows for multiple configurations for both monolithic and pipeline DS, including state-of-the-art models for the NLU,

NLG and DST components. Listing 3.1 shows an example, written in the Python programming language, of how to create a simple DS instance using Convlab-2. With the purpose of reducing the impact of potential upstream (NLU) and downsteam (NLG) errors, the DS pipeline used in this work has no NLU module (as dialogues are be already encoded into dialogue acts in the library), a simple template NLG module (where utterances are generated from user and system dialogue acts by using pre-defined templates) and a simple rule-based DST for the System.

```
from convlab2.dialog_agent.agent import PipelineAgent
from convlab2.dialog_agent.env import Environment
from convlab2.dst.rule.multiwoz import RuleDST
from convlab2.nlg.template.multiwoz import TemplateNLG
from convlab2.evaluator.multiwoz_eval import MultiWozEvaluator


# ------- DS instance -------
# --- user ---
dst_usr = None
policy_usr = RulePolicy(character='usr')
user_nlg = TemplateNLG(is_user= True, mode='manual')

# --- system ---
dst_sys = RuleDST()
policy_sys = PPO()
sys_nlg = TemplateNLG(is_user = False, mode='manual')

# assemble
user_simulator = PipelineAgent(None, None, policy_usr, None, 'user
    ')

evaluator = MultiWozEvaluator()

env = Environment(None, user_simulator, None, dst_sys, evaluator)
```

Listing 3.1: DS instance in Convlab

ConvLab-2 comes with a user simulator, and allows for easy set up and experimentation with various DS approaches. Specific focus was given to Reinforcement Learning methods, for which ConvLab-2 provides a specialised training environment. To train a system policy via Reinforcement Learning, the environment module can be used to connect the the agent – the user simulator – and the system. In particular, the environment in Convlab-2 follows a similar design to OpenAI Gym ([7]), implementing the following methods:

- `env.reset()` to sample an initial belief state, which contains the (randomly selected) initial user goal;

- `next state, reward, done = env.step(action)` for taking steps and retrieving the immediate reward, the next state where the MDP jumps to, and whether the dialogue has been finished.

Moreover, the evaluator module can be called within the environment to assess the dialogue success, which is independent of the environment variable `done`[2]. The reward function is designed to provide a reward of -1 for every environment step where the dialogue is not successful (`done = False`), and a positive reward of L when the dialogue has successfully finished (`done = True` and `evaluator.task success = True`), where L is the maximum number of turns in every dialogue instance, and is set to $L = 40$ in Convlab-2.

For every step, the environment has to receive the actions to take, which are the output of the Policy module (for example, a Proximal Policy Optimisation algorithm as shown in Listing 3.1). The Policies that are included in Convlab-2 are:

- a Rule-based policy, where for every user dialogue act there are handcrafted system responses;

- a Maximum Likelihood Estimation (MLE) policy that learns the state-action pairs directly from the dataset by imitation;

- three RL policies: REINFORCE ([71]), Proximal Policy Optimization (PPO, [53]), and Guided Dialog Policy Learning (GDPL, [78]).

Convlab-2 also includes a module called the analyzer, that can be used to compute the following performance metrics:

- Complete rate: the percentage of dialogues completed over the total of dialogues, where completed means that all information requested by the user was given by the system, independently of it correctness:

[2] A dialogue can be successful or not even when it has reached an end, where `done = True`

$$\texttt{Complete rate} = \frac{\texttt{completed dialogues}}{\texttt{total dialogues}}$$

- Success rate: the percentage of successful dialogues over the total of dialogues, where success means that all the user's requests were correctly fulfilled by the system:

$$\texttt{Success rate} = \frac{\texttt{successful dialogues}}{\texttt{total dialogues}}$$

- Book rate: the percentage of successful bookings over the total of (bookable)[3] dialogues:

$$\texttt{Book rate} = \frac{\texttt{successful bookings}}{\texttt{total of bookable dialogues}}$$

Also, it is useful to consider the average number of interactions (turns) between the system and the user as another performance metric (the lower, the better).

Similar to [78], I use PPO as the main RL baseline algorithm, given that it is a standard policy gradient algorithm that can easily be extended to include intrinsic motivation approaches. The remaining sections describe the PPO algorithm and the intrinsic motivation approaches that I use in this work.

3.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO, [53]) follows a similar idea as Trust Region Policy Optimization (TRPO, [51]), where the policy parameter update is controlled in such a way that the policy does not change too much with every optimisation step. In TRPO, this is achieved by enforcing the KL divergence between the updated policy and the old one to be limited by an upper bound δ:

$$\mathop{\mathbb{E}}_{s\sim\rho^{\pi_{\text{old}}}} \left[D_{\text{KL}} \left(\pi_{\theta_{\text{old}}}(\cdot \mid s) \| \pi_{\theta}(\cdot \mid s) \right] \leq \delta \right. \tag{3.1}$$

[3] Note that not all user goals necessarily impose a booking, e.g. request for an address or postcode.

This constraint is used to maximise the objective (known as the surrogate function) as:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \right] \right] \leq \delta \end{aligned} \tag{3.2}$$

where the surrogate function $\frac{\pi\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}t$ comes from the gradient estimator following the Policy Gradient theorem, $\hat{g} = \hat{\mathbb{E}}t \left[\nabla\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$. Instead of solving the constrained problem in 3.2, [51] proposes to solve the unconstrained problem:

$$\underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}t - \beta \text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t) \right] \right] \tag{3.3}$$

for some coefficient β. In practice, experiments show that it is hard to choose a fixed coefficient β that performs well during learning. In fact, TRPO can lead to unstable policy updates where the ratio $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ between the updated policy and the old one can become big. Therefore, a simplified maximisation problem is proposed in [53], where a constraint is imposed over the ratio $r(\theta)$ to remain within an interval $[1-\varepsilon, 1+\varepsilon]$, ε being an hyperparameter. Therefore, the surrogate function in PPO becomes:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(rt(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t \right) \right] \tag{3.4}$$

In practice, the policy $\pi(\theta)$ can be estimated using an Actor Critic architecture, therefore the surrogate function is optimised jointly with the value function. Additionally, an Entropy term can be included to provide enough exploration:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{3.5}$$

where c_1 and c_2 are hyperparameters of the model.

The estimated advantage function $\hat{A}_t$, defined as

$$\hat{A}t = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^V = -V(s_t) + \sum_{l=0}^{\infty} \gamma^l r_{t+l} \tag{3.6}$$

where $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$, can be computed following a similar approach

to TD(λ) known as generalised advantage estimation (GAE, [52]):

$$\hat{A}t = \delta t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{3.7}$$

The pseudocode for PPO using GAE and AC is shown in Algorithm 1.

Algorithm 1: PPO

Initialize policy parameter θ_0 and value function parameter ϕ_0;
for $i = 0, 1, 2, \ldots$ **do**
  Run current policy π_{θ_i} until N timesteps are obtained;
  Compute δ_t^V at all timesteps $t \in \{1, 2, \ldots, N\}$, using $V = V_{\phi_i}$;
  Compute $\hat{A}t = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^V$ at all timesteps;
  Optimise surrogate objective $L_t^{CLIP+VF+S}(\theta)$ w.r.t. θ, as in Equation 3.5;
  $\theta_{\text{old}} \leftarrow \theta$
end

3.4 Intrinsic Motivation

While in classic RL algorithms, such as DQN and PPO, exploration is improved by different *ad-hoc* techniques such as ε-greedy or Entropy bonus (as in Equation 3.5), there exist more sophisticated exploration methods that allow for a faster and more stable learning. [6] proposes a pseudo-count model where the visiting frequency of states is estimated through a density model. [60] extend this idea to high dimensional spaces by using a hash mapping function to encode the states and make it possible to keep the count of visiting events. On the other hand, other approaches ([50], [39], [4]) introduce the concept of intrinsic motivation, which is a measure of the novelty of each visited state during learning. This means that less frequently visited states become more novel, adding some form of reward bonus (usually proportional to a prediction error) that allows to enforce exploration. In [38] intrinsic motivation is computed as the information gain, defined as the KL divergence of a prior distribution from its posterior.

A well-known limitation of classical RL algorithms is reward sparsity, where exploration becomes harder due to the lack of instant feedback from the environment for every learning step. One approach that has been used to overcome this

limitation is estimating intrinsic rewards, such that the total reward is computed as the sum of these intrinsic rewards and the ones provided by the environment (extrinsic rewards). In [47], the authors introduce the concept of episodic memory, where large intrinsic reward states are appended so that novel states will be sampled during learning. In [41] the concept of Curiosity-driven learning is introduced, where an intrinsic curiosity module, composed by a forward and an inverse estimation models, is used to generate step-wise intrinsic rewards. A different approach that follows the same idea of intrinsic rewards as a "novelty magnitude" is used in [10], where the technique of Random Network Distillation (RND) as an exploration strategy is used. RND encourages visiting novel states by measuring the prediction error of a fixed, randomly initialised neural network at every visited state. RND has been recently used in [3], which in conjunction with an episodic memory that generates an intra-episode intrinsic reward as in [47], achieves state-of-the-art results in the human Atari benchmark ([5]). An extensive review of intrinsic motivation methods in RL can be found in [2].

In this work I use intrinsic curiosity (IC) and RND to improve the exploration in DS policy learning, therefore both of these methods are described more extensively as follows.

3.4.1 Curiosity-driven Reinforcement Learning

In curiosity-driven RL, the policy learns to optimise the extrinsic reward r_t^e given by the environment plus an intrinsic reward r_t^i generated by the Intrinsic Curiosity Module (ICM, see Figure 3.3), that is learned in a self-supervised way. The ICM is composed of two sub-modules: the forward and inverse dynamics models. It takes raw states s_t (or observations) from the environment and encodes them into a feature vector $\Phi(s_t)$, both for current state s_t and next state s_{t+1}, and also takes as input the agent's action a_t. The forward model is trained on $(\Phi(s_t), a_t)$ to output an estimate of the (featurised) next states, $\hat{\Phi}(s_{t+1})$, where its target loss function L_F is

$$L_F\left(\phi\left(s_t\right), \hat{\phi}\left(s_{t+1}\right)\right) = \frac{1}{2}\left\|\hat{\phi}\left(s_{t+1}\right) - \phi\left(s_{t+1}\right)\right\|_2^2 \tag{3.8}$$

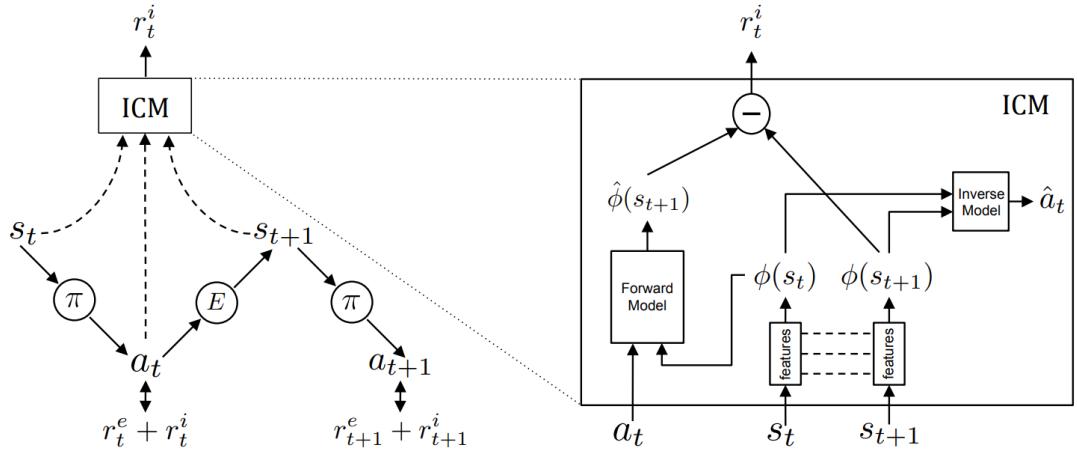

Figure 3.3: Reinforcement Learning with Intrinsic Curiosity

The inverse model takes the encoded feature vectors $\Phi(s_t)$ and $\Phi(s_{t+1})$ and is trained to estimate the input action, $\hat{a}_t = g(s_t, s_{t+1}; \theta_I)$, where θ_I is learned by optimising the loss function:

$$\min_{\theta_I} L_I(\hat{a}_t, a_t) \tag{3.9}$$

The absolute value of the difference between the predicted feature vector of the next state $\hat{\Phi}(s_{t+1})$ and the feature vector of the actual next state $\Phi(s_{t+1})$ is used as the intrinsic reward r_t^i signal:

$$r_t^i = \frac{\eta}{2} \left\| \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \right\|_2^2 \tag{3.10}$$

for a constant $\eta > 0$. Therefore, the joint optimisation problem becomes

$$\min_{\theta_P, \theta_I, \theta_F} \left[-\lambda \mathbb{E}_{\pi(s_t; \theta_P)} [\Sigma_t r_t] + (1-\beta) L_I + \beta L_F \right] \tag{3.11}$$

where λ is a hyperparameter that indicates the overall contribution of the return expectation to the total loss function. A pseudocode for IC is shown in Algorithm 2.

Algorithm 2: IC

Randomly initialize encoder, forward and inverse networks parameters;
Get initial state s_0;
Set η and α;
for $i = 0, 1, 2, \ldots$ **do**
  Sample transition a_i, s_{i+1} given s_i;
  Compute embeddings $\phi(s_i)$ and $\phi(s_{i+1})$;
  Optimise the forward (Equation 3.8) and inverse (Equation 3.9) loss by gradient descent;
  Compute intrinsic reward r_i^{int} as in Equation 3.10;
  $\eta \leftarrow (1-\alpha)\eta$
end

3.4.2 Random Network Distillation

In [3], when the environment takes a step based on a policy action and transitions into the next state, RND computes an exploration bonus that is based on predicting the output of a fixed and randomly initialized neural network on the next state, given the actual next state of the MDP. Two neural networks are used: a target network that is randomly initialised and fixed during training, and a predictor network that is trained on the agent observations. The target network maps the next state (or observation in general) into an embedding $f : \mathbb{S} \to \mathbb{R}^k$, and the predictor network into a different embedding $\hat{f} : \mathbb{S} \to \mathbb{R}^k$. Then, the predictor network is trained to minimise the expected prediction error, $\|\hat{f}(\mathrm{x};\theta) - f(\mathrm{x})\|^2$, by gradient descent. The prediction error is therefore expected to be higher for novel states than the ones the predictor network has seen during training. Moreover, the prediction error can be used (up to some domain-specific scale) as an intrinsic, step-wise reward. A diagram of RND is shown in Figure 3.4 [4], and the pseudocode in Algorithm 3.

While both RND and IC can be trained jointly with the policy combining intrinsic and extrinsic rewards, it is also possible to modify these algorithms to separate the intrinsic reward estimation from the policy learning. In particular, the intrinsic reward estimation problem is very different to the policy learning problem,

[4] Source: `https://openai.com/blog/ /reinforcement-learning-with-prediction-based-rewards/`

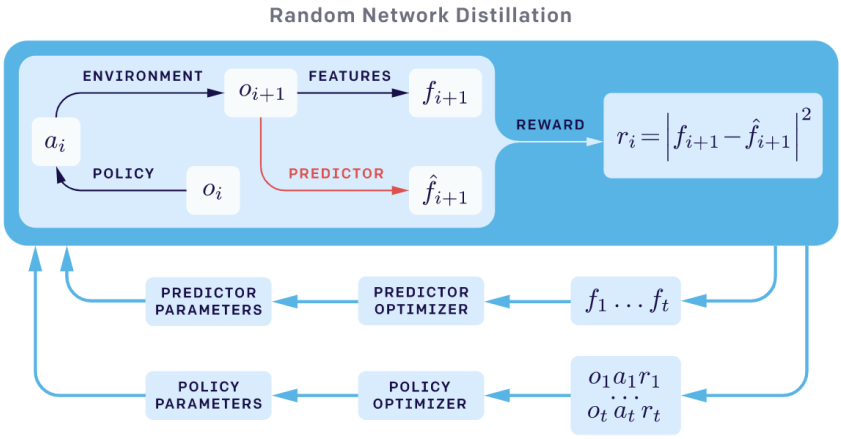

Figure 3.4: Random Network Distillation

Algorithm 3: RND

Randomly initialize target and predictor networks parameters, θ_0' and θ_0;
Freeze θ_0';
Get initial state s_0;
Set η and α;
for $i = 0, 1, 2, \dots$ **do**
  Sample action a_i given s_i;
  Compute embeddings $f_{\theta_i}(s)$ and $f_{\theta_i'}(s_i)$;
  Optimise the loss $\|f_{\theta_i'}(s_i) - f_{\theta_i}(s)\|^2$ w.r.t. θ by gradient descent;
  Compute intrinsic reward $r_i^{int} = \eta \|f_{\theta_i'}(s_i) - f_{\theta_i}(s)\|^2$;
  $\eta \leftarrow (1-\alpha)\eta$
end

therefore for IC it makes more sense to use two different learning rates. This will be discussed in the next section. It is also worth noticing that in both RND and IC the intrinsic rewards are a measure of the distance in some latent space, which ultimately is proportional to the novelty of each transition in the environment.

Chapter 4

Experimental Setup

4.1 Environment

As mentioned in the Methodology section, the environment that is used in this work corresponds to Convlab-2 ([77]). This open-source library was initially released around February 2020, and has been updated and improved since then. Moreover, several modifications including changes in the user agent, NLG, goal generators and models were made during the course of this project. To have a stable code-base while carrying out experiments, and also to avoid having to constantly adjust to the aforementioned changes, Convlab-2 was frozen at commit before **bdc9dba** (inclusive).

The training and evaluation of the models is carried out without distinguishing between test and evaluation data, as is the general case for any RL work.

4.1.1 Belief State Encoder

Convlab-2 environment states and agent observations are in fact raw *belief states*, where the DST keeps track of the user goals and dialogue acts history of each dialogue instance. While these belief states are structured into dictionaries, a belief state encoder[1] is implemented to translate from raw belief states to fixed-length arrays. Also, Convlab-2 agent's raw actions are the concatenation of different dialogue acts, therefore the belief state encoder is also used to translate them into fixed-length arrays. This information is summarised in Table 4.1, where the output

[1] Note: do not confuse this with the state encoders in the ICM.

Input (structure)	Output (structure, dimension)
Belief state (dict)	(*array*, 340)
Action (dict)	(*array*, 209)

Table 4.1: Belief State and Action encoder input and outputs.

Hyperparameter	Value
Number of training steps	1M
Discount factor (γ)	0.99
Dialogue sample batch size	32
Optimiser batch size	32
Optimiser model	AdamW ([31])
Activation function	ReLU

Table 4.2: Hyperparameters shared by all models

arrays dimensions are shown.

4.2 General Model Hyperparameters

Besides model-specific hyperparameters, there are hyperparameters that are common for all models trained in this work, which are listed in Table 4.2. The choices for batch size and discount factors follow commonly used values and were not tested during hyperparameter tuning. The number of training steps was fixed to 1 million for all experiments due to training wall time and limited resources, as will be discussed in the next section.

4.3 Models Implementation

4.3.1 PPO

The implementation of the PPO algorithm follows the one included in Convlab-2, where only the hyperparameters were changed[2] after studying the learning performance with different learning rates, optimiser models and algorithm-specific hyperparameters. In this implementation there are two neural networks that represent the Actor Critic architecture, therefore, since there are no shared parameters, two

[2]This means, the PPO hyperparameters used in this work are different than the ones suggested by Convlab-2 as the optimal ones. I found that with these reported hyperparameters, PPO converges and remains stable during training, while using the hyperparameters suggested by Convlab-2 the algorithm is not able to learn.

Hyperparameter	Value
Actor input dimension	340
Actor hidden dimension	100
Actor output dimension	209
Learning rate	5e-6
Surrogate clip (η)	0.1
GAE λ	0.95

Table 4.3: Hyperparameters for PPO (Actor)

Hyperparameter	Value
Critic input dimension	340
Critic hidden dimension	50
Critic output dimension	1
Learning rate	1e-5

Table 4.4: Hyperparameters for PPO (Critic)

different optimisers are used. Tables 4.3 and 4.4 show the hyperparameters for the Actor and Critic models of PPO. During training, for each sampled batch of dialogues the same policy is trained for five iterations, and the gradient for the Actor loss is clipped to a value of 1. The entropy bonus is ignored ($c_2 = 0$ in Equation 3.5) so that all exploration will be given by the intrinsic motivation models. All these PPO hyperparameters are kept fixed for training the RND and IC models.

4.3.2 Random Network Distillation

The implementation of the RND model follows Algorithm 3. There are two modes for this model to work, either taking the dialogue acts between the user and the system to compute step-wise intrinsic rewards (which will be called **DAs**, e.g. *RND (DAs)*), or using the (natural language) utterances instead (which will be called **Utt**, e.g. *RND (Utt)*). Both modes of RND are warmed-up and then jointly trained along the policy.

4.3.2.1 Random Network Distillation from Dialogue Acts

When the dialogue acts are used to produce intrinsic rewards, two feed-forward neural networks are used for the predictor and target models (where the latter has fixed parameters during training). The hyperparameters for these networks are shown in

Hyperparameter	Value
Input dimension	418
Hidden dimension	524
Output dimension	340

Table 4.5: Hyperparameters for RND (DAs) predictor and target networks

Hyperparameter	Value
Language model	BertForSequenceClassification
Tokenizer	bert-base-uncased
Input dimension (max)	512
Output dimension	340

Table 4.6: Hyperparameters for RND (Utt) predictor and target networks

Table 4.5.

4.3.2.2 Random Network Distillation from Utterances

When utterances are used to produce intrinsic rewards, a pre-trained BERT model([13])[3] is used for both predictor and target. While it is possible to fine-tune the entire pre-trained BERT model over the course of training, I investigated the effect of freezing all BERT layers except for the last one (the classification head), which is trained jointly with the policy. As the results for both cases are similar but the training wall time for the former is much bigger, I opted for the approach of training only the classifier head of BERT for all cases. The hyperparameters for both predictor and target models based on BERT are shown in Table 4.6.

4.3.2.3 Random Network Distillation General Hyperparameters

Table 4.7 shows the hyperparameters for RND used in this work, including both modes using DAs and utterances. It should be noted that some hyperparameters are different for RND (DAs) and RND (Utt), as all of them are the result of a careful grid search-based hyperparameter tuning.

4.3.3 Intrinsic Curiosity

Similar to the case of RND, IC is implemented to be trained in two modes: *IC (DAs)*, where the tuple (state, action, next state) is taken as a dialogue act by the

[3] A *BertForSequenceClassification* from Hugging Face is used (`https://huggingface.co/transformers/model_doc/bert.html#bertforsequenceclassification`)

Hyperparameter (mode)	Value
Learning rate (both)	1e-3
Warmup episodes (DAs)	100
Warmup episodes (Utt)	200
Moving average steps period (DAs)	2
Moving average steps period (Utt)	10
Update rounds (DAs)	5
Update rounds (Utt)	1
RND gradient clip value	10
r_t^{int} multiplier initial value (DAs)	1.0
r_t^{int} multiplier initial value (Utt)	5.0
r_t^{int} multiplier final value (both)	0.001
r_t^{int} linear annealing steps (DAs)	20000
r_t^{int} linear annealing steps (Utt)	50000

Table 4.7: Hyperparameters for RND (DAs and Utt) training.

ICM, and *IC (Utt)* where instead the ICM takes NLG-generated utterances from the tuple dialogue acts.

Following Algorithm 2, IC can also be trained by either jointly optimising the ICM with the policy, or by pre-training the ICM and then keeping it fixed to use it to generate intrinsic rewards. While the former method follows the original idea as in [41], it limits the joint model to be trained using a single optimiser and therefore it does not take into account that IC and the policy are different problems, therefore requiring different learning hyperparameters (e.g. different learning rates). As this will be discussed later in the Results section, hyperparameters shown here correspond to the pre-training case.

The forward and inverse models are implemented as feed-forward neural networks using intermediate ReLU activations (see Table 4.8 for their dimensions). In IC (DAs) the state encoder is a two layer feed-forward neural network that takes (encoded) dialogue acts and maps them into embeddings, while in IC (Utt) the encoder itself is a BERT model that uses a BERT base uncased tokenizer, similar to RND (Utt). When the pre-training strategy is used, the ICM is trained for a fixed number of environment steps and then the loss of the forward model (scaled by a hyperparameter η) is used as the intrinsic reward signal.

Hyperparameter	Value
Number of steps for pre-train	1000
Learning rate (pre-train)	1e-3
Learning rate (joint)	1e-5
Update rounds	1
Gradient clipping parameter	10
η	0.01
$\beta_{\texttt{DAs}}$	0.2
$\beta_{\texttt{Utt}}$	0.1
$\beta_{\texttt{joint}}$	0.8
$\lambda_{\texttt{pol}}$	0.5
Inverse model hidden dimension	524
Forward model hidden dimension	524
State encoder output dimension	256
State encoder max length	200

Table 4.8: Hyperparameters for IC (DAs and Utt) model

4.4 Evaluation of Trained Policies

While in its current state Convlab-2 does not allow access to the model performance during training, I wrote a wrapper[4] for the Analyzer module to be able to log and analyse performance over the course of training. There is one hyperparameter in this wrapper that correspond to the number of sampled dialogues to be used in order to get average metrics. Naturally, training and evaluating simultaneously leads to an increase in wall time. On the other hand, having access to evaluation results during training becomes crucial to do hyperparameter tuning. However, the number of dialogue samples for evaluation has to be taken into account: while evaluating for a small number of sample dialogues (e.g. 50) decreases the training time, it results in unreliable indications of how the model is truly performing. Therefore, all evaluations in this work are carried out with 1000 sampled dialogues from the environment, so that the results shown in the learning curves are consistent with the follow-up performance metrics breakdown.

[4]See `https://github.com/thenickben/ucl_thesis/utils.py`

Chapter 5

Results and Discussion

5.1 Baseline Models

Convlab-2 has four RL algorithms implemented for its System Policy module, and it reports[1] the (average)[2] success rate as shown in Table 5.1.

Model	Success rate
MLE	0.56
REINFORCE	0.54
PPO	0.74
GDPL	0.58

Table 5.1: Success rate for baseline models in Convlab-2

It should be noted that while GDPL ([59]) is referred to as the state-of-the-art model in terms of performance on the MultiWOZ dataset in [77], the authors of Convlab-2 report a success rate of 0.74 for PPO and just 0.58 for GDPL, marginally above MLE and REINFORCE. Moreover, the reported success rate in [59] (where the same dataset was used) is 0.59. This suggests that the baseline training hyper-parameters suggested by Convlab-2 are not optimal, or there may be bugs in some components of the library.

[1] As indicated these results correspond to the version up to the commit **bdc9dba** (inclusive), see `https://github.com/thu-coai/ConvLab-2#Module-Performance-on-MultiWOZ`

[2] I will refer to any performance metric by its name (e.g. Book rate), but it should be noticed that I am always refering to an average over a given number of sampled dialogues.

5.2 Effect of the Number of Sampled Dialogues in Evaluation

In this section I justify empirically the qualitative intuition discussed in Section 4.4 of fixing the number of dialogue samples (n_{eval} for short) to evaluate any trained policy. Figure 5.1 shows the Success rate and its standard deviation when a PPO model, trained for 1M steps, is evaluated while varying n_{eval}. For each n_{eval}, 20 computations of Success rate were performed. Results indicate that setting $n_{eval} = 1000$ lead to a very small variation of the performance metrics. Thus, in order to get a highly reliable estimation of model performance during training, I set $n_{eval} = 1000$ for all experiments.

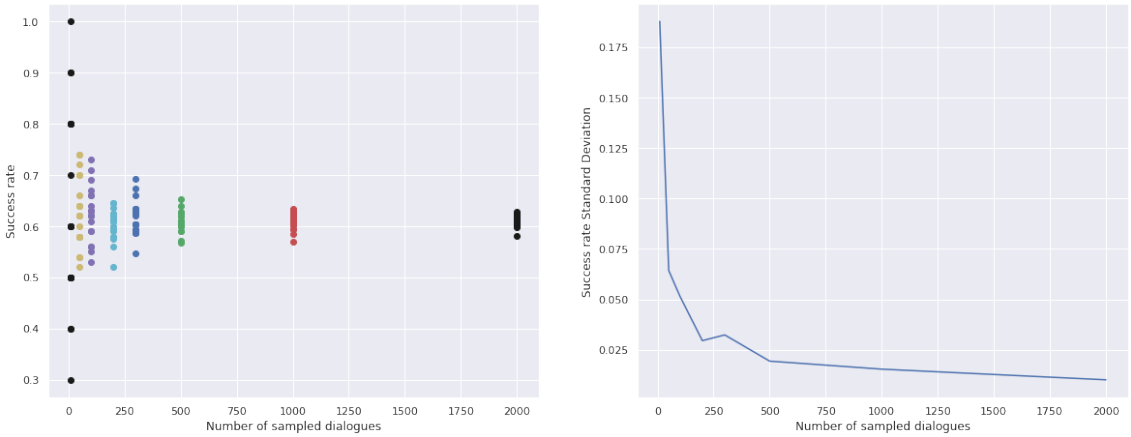

Figure 5.1: Effect of the number of sampled dialogues in evaluation

5.3 IC+PPO Joint Optimisation

As discussed in the previous section, the Intrinsic Curiosity model can be trained jointly with PPO so that for every step taken in the environment, the ICM will be optimised and its intrinsic rewards will be adjusted to all the information that is seen during training. However, this poses two problems: on the one hand, it is not reasonable to assume that the learning rate for the ICM will be the same as for PPO. For example, if the learning rate is set to a very small value, the ICM will hardly learn, yielding the same level of intrinsic rewards throughout training, and if the learning rate is too high, the PPO policy may not converge. On the other hand, I

experimented with joint optimisation both for DAs and utterances, and found that while for the former the training seems stable, for the latter the training time rises considerably (see Figure 5.2). Moreover, results obtained with the joint models are similar to the ones obtained by first pre-training the ICM and keeping it fixed during policy learning. Results for IC+PPO (that will be named as *IC joint*) for dialogue acts will be also shown in the following subsections.

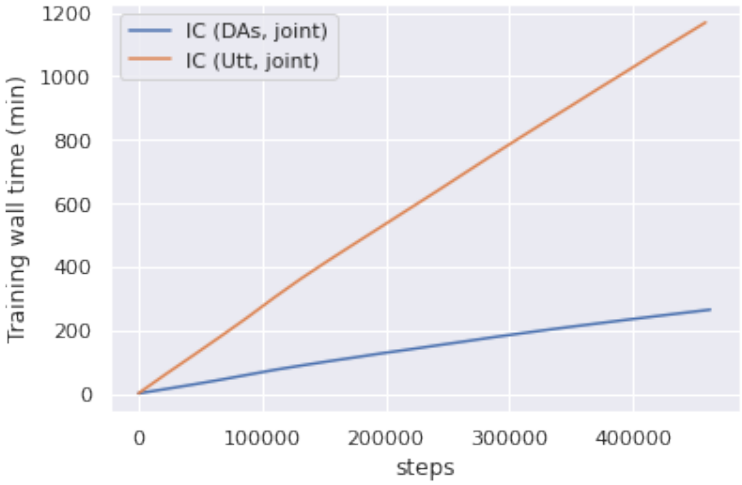

Figure 5.2: Training wall time for IC+PPO joint training

5.4 Effect of MLE Pre-training Epochs

As suggested in [77], a widely used technique in DS is to pre-train RL policies by imitating the dialogues in the dataset (MLE). However, the number of epochs[3] used to pre-train the policy can impact the subsequent training results. Therefore, I carried out an analysis where I trained a base PPO model using five different numbers of pre-training epochs (see Figure 5.3), and found that the optimal number of epochs such that the base policy is able to converge is between 24 and 32. Hence I chose using 32 epochs for MLE pre-training for all the experiments in this work.

[3]Here one epoch corresponds to a full dialogue rollout (e.g. 40 dialogue steps).

Figure 5.3: Learning curves for different MLE pre-train epochs.

5.5 Learning Curves

Figures 5.4, 5.5, 5.6 and 5.7 show the average success rate, book rate, number of turns and rewards, respectively, during training[4]. It can be seen that 1M environment steps are enough for all the models to reach convergence. In Figure 5.4, RND, trained using both dialogue acts and utterances, shows the fastest increase in success rate at the beginning of training (around 100k steps), while the remaining models grow at a similar rate than the baseline PPO. However, after 0.5M steps almost all models except IC (DAs) get higher success rates than PPO, and all of them achieve this by the end of the training at 1M steps. RND from utterances gets a success rate over 12 percent above PPO, which is a substantial improvement given the complexity of this environment. Moreover, it becomes evident that intrinsic motivation models trained in the semantic space – that is, generating intrinsic rewards directly from the utterances – get better results than when they are trained in the dialogue acts space. This can be explained by noticing that for one single dialogue act there are many different utterances that can be generated, creating a more granular level

[4]Results shown are the average over three different random seeds.

of intrinsic rewards which results in an overall improved exploration. Similar results are obtained for the average book rate in Figure 5.5, where intrinsic models trained from utterances surpass the baseline. However, the highest book rate is achieved by IC joint (DAs), marginally above RND (Utt). For the average turns, it can be observed in Figure 5.6 that both RND (Utt) and IC (Utt) achieve the lowest number, around 8.2 turns per dialogue which is a substantial improvement over the 10 turns per dialogue for the case of PPO. In general, policies that use a lower number of turns for completing a (successful) dialogue can be said to have a more efficient interaction with the user (e.g. achieving a goal with fewer interactions). Finally, the average rewards during training in Figure 5.7 show how the intrinsic motivation models introduce a higher variance in the total rewards compared to PPO at the beginning of training, and achieve higher average rewards with less variance by the end.

One important remark about shown learning curves is that, if instead of training the models until convergence (or for a fixed number of steps), the best performing models are chosen, then PPO would become comparable (with marginally a lower performance) to the intrinsic motivation models[5]. This ultimately depends on the motivation for training DS: for industrial purposes it may be reasonable to select the best performing model (independently of its convergence properties), while for academic purposes it becomes more important to develop algorithms that can show a stable and convergent behaviour during learning.

5.6 Performance Metrics

A performance evaluation of the final models, after training for 1M steps, was carried out using the analyzer module provided by Convlab-2, where a number of sampled dialogues are used to compute the complete rate, success rate, average number

[5]In fact, this may be the reason why Convlab-2 reports a success rate of 0.72. As evidenced in its training script, it saves the model with the highest success rate.

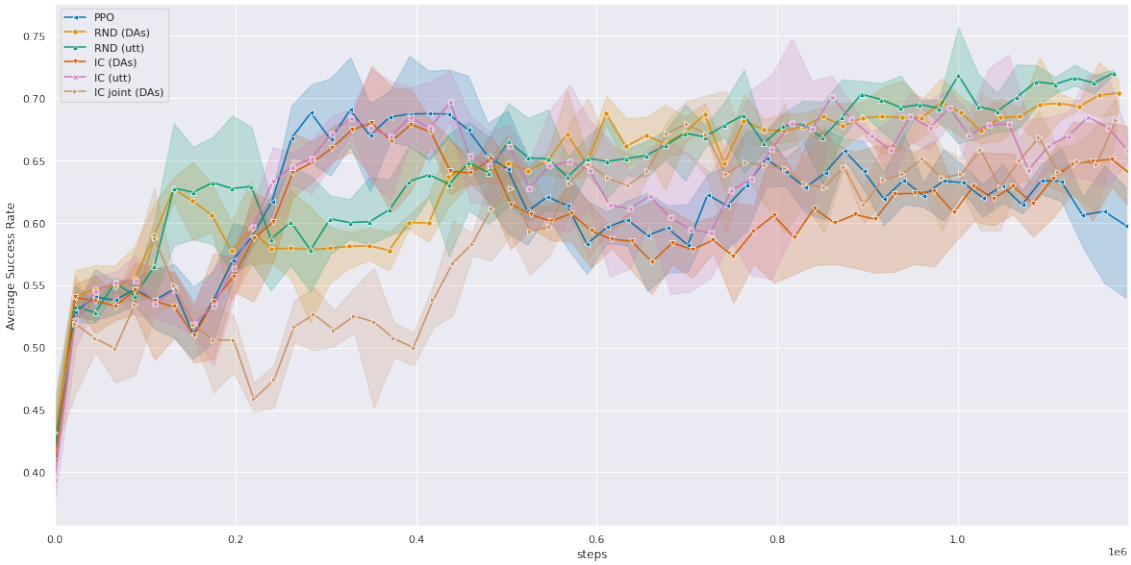

Figure 5.4: Average success rate during training.

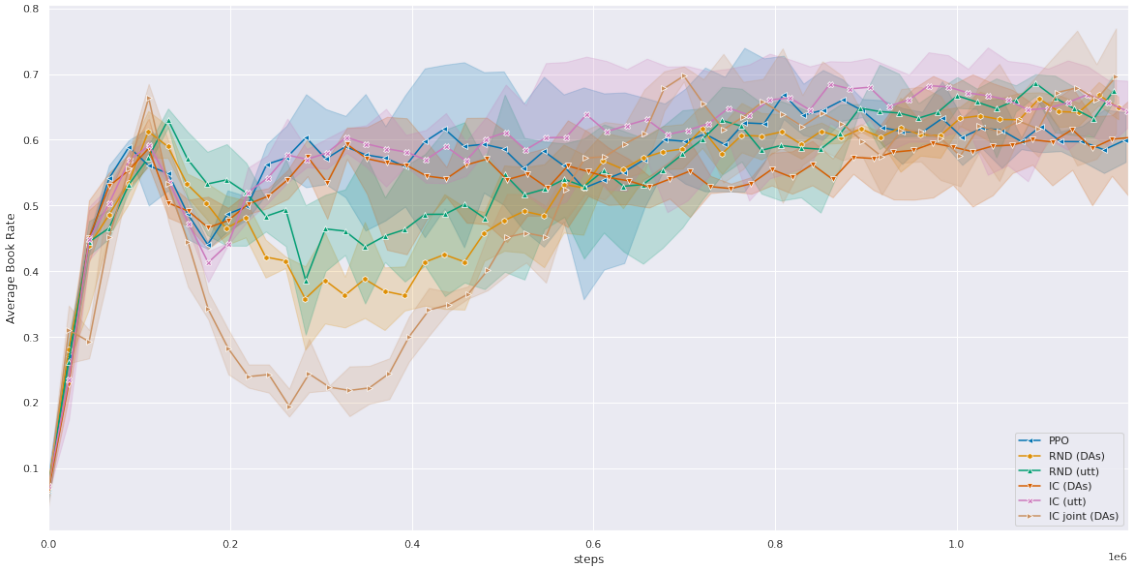

Figure 5.5: Average book rate rate during training.

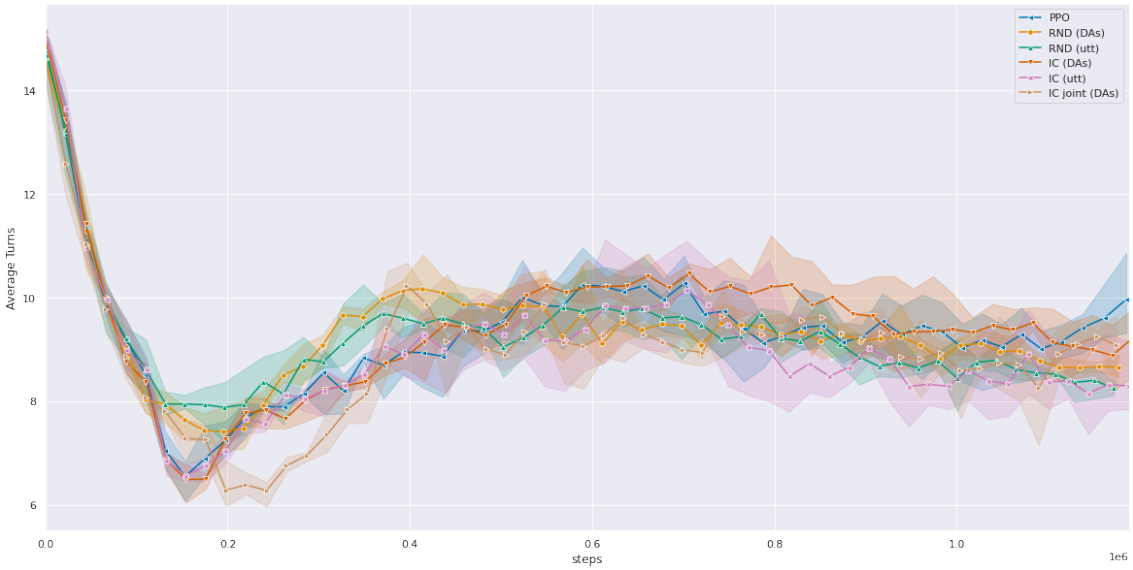

Figure 5.6: Average turns during training.

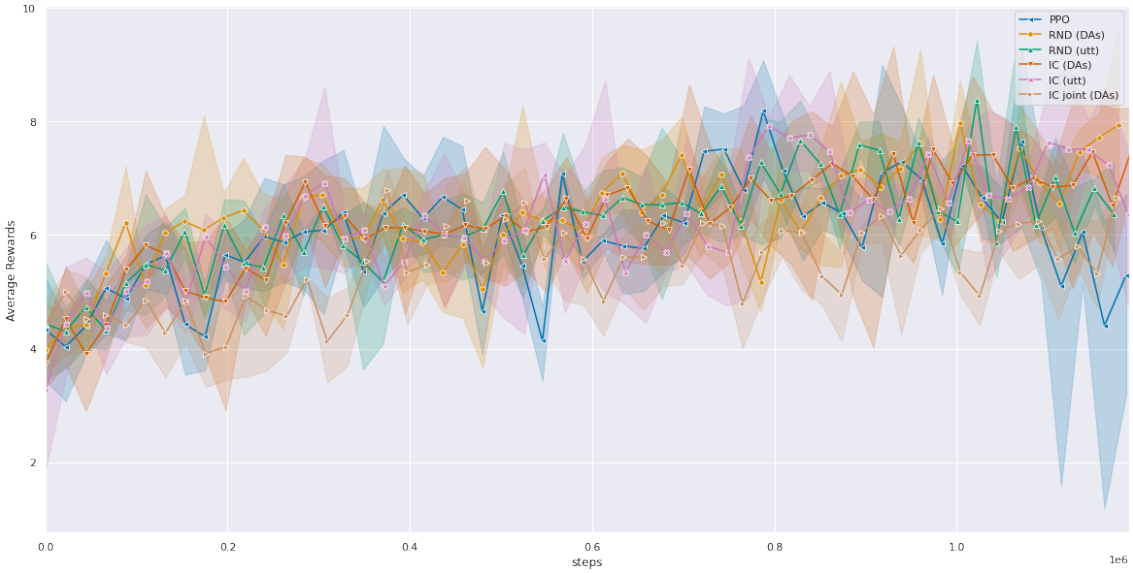

Figure 5.7: Average rewards during training.

of turns[6], and book rate. Table 5.2 shows all these metrics for the baseline PPO and the intrinsic motivation models[7]. As in Figures 5.4, 5.5 and 5.6, it is evident that RND (Utt) surpass the success rate of PPO by 13 percent, which is a considerable improvement over the baseline. Moreover, it also shows improvements in all the other performance metrics. Intrinsic curiosity models also improve over the baseline, however this improvement is moderate compared to RND[8].

	Complete	Success	Turn	Book
PPO	0.72	0.60	19.65	0.59
RND (DAs)	0.78	0.67	17.66	0.612
RND (utt)	**0.82**	**0.73**	**16.80**	**0.68**
IC (DAs)	0.74	0.64	17.98	0.57
IC (utt)	0.76	0.65	17.67	0.60
IC joint (DAs)	0.78	0.65	18.37	**0.69**

Table 5.2: Average Performance Metrics

Convlab-2 also allows to break down the performance by domains, where it takes into account which domains contained in each sampled initial user goal were correctly addressed by the system. Performance results per domain are shown in Table 5.3, where domains are shown in decreasing order of their probability of being sampled in MultiWOZ (see also Figure 3.1).

	PPO	RND (DAs)	RND (utt)	IC (DAs)	IC (utt)	IC joint (DAs)
Restaurant	0.72	0.77	0.78	0.77	**0.82**	0.75
Hotel	0.74	0.81	**0.82**	0.75	0.79	0.70
Train	0.66	0.73	0.73	0.66	0.58	**0.81**
Attraction	0.84	0.87	**0.96**	0.93	0.88	0.89
Taxi	0.63	0.97	**0.99**	0.96	0.93	0.85
Hospital	1.00	1.00	1.00	1.00	1.00	1.00
Police	1.00	1.00	1.00	1.00	1.00	1.00

Table 5.3: Success rate per domain.

The low probability of sampling user goals that include either hospital or police domains in addition to the simplicity of these goals (where the user only requests

[6]The reported number of turns by Convlab-2 counts every interaction as one step, while results shown in Figure 5.6 counts a question-answer pair as one dialogue turn.

[7]Shown numbers are the result of evaluating three trained models (for three different random seeds) for 1000 dialogues, and averaging their results.

[8]The only exception is the book rate for IC joint (DAs), which is almost similar as for RND (Utt).

for simple information or a telephone) makes all models have perfect scores. RND (Utt) gets the biggest success rate for hotel, which is the second most frequent domain in the database, and also for attraction and taxi (in the latter its success rate is almost 100 percent), while IC shows a marginal better metric for restaurant (IC (Utt)) and for train (IC joint (DAs)). It should be noted that in order to relate these results to the ones in Table 5.2, the distribution of the number of domains (see Figure 3.2). Moreover, it should be taken into account that the goal generator module in Convlab-2 lacks a representative distribution of the different domain combinations that can be sampled from MultiWOZ, e.g. it never samples hospital and taxi in the same user goal.

5.7 Sensitivity to the Number of Domains

As seen in 5.3, simple domains such as hospital or police lead to better system performance than more complex domains that include multiple requests composed by information and bookings. Given that MultiWOZ is a multi-domain dataset, assessing the performance of the system with respect to the number of domains in sampled user goals allows a direct comparison of the robustness and scalability of intrinsic motivation-based policies compared to the baseline. Figures 5.8 and 5.9 show the success rate and book rate, respectively, given the number of domains for 100 sampled dialogues. It can be seen that intrinsic motivation models are more robust to the number of domains covered by a user goal than the baseline PPO, where RND (Utt) in particular keeps an average success rate of around 70 percent for two and three domains, while PPO quickly decreases to less than 30 percent for three domains[9]. The sensitivity of the book rate with respect of the number of domains presents an interesting behaviour, where it decreases going from one to two domains, but increases from two to three [10]. Overall, intrinsic motivation models are again more robust than PPO in terms of book rate, which suggests that adding semantic-driven intrinsic rewards can enable a Dialogue System to handle

[9] Similar results for PPO were found in [59].

[10] A similar shape was found for one of the baseline models in [59], however their results for PPO do not match the ones in this work.

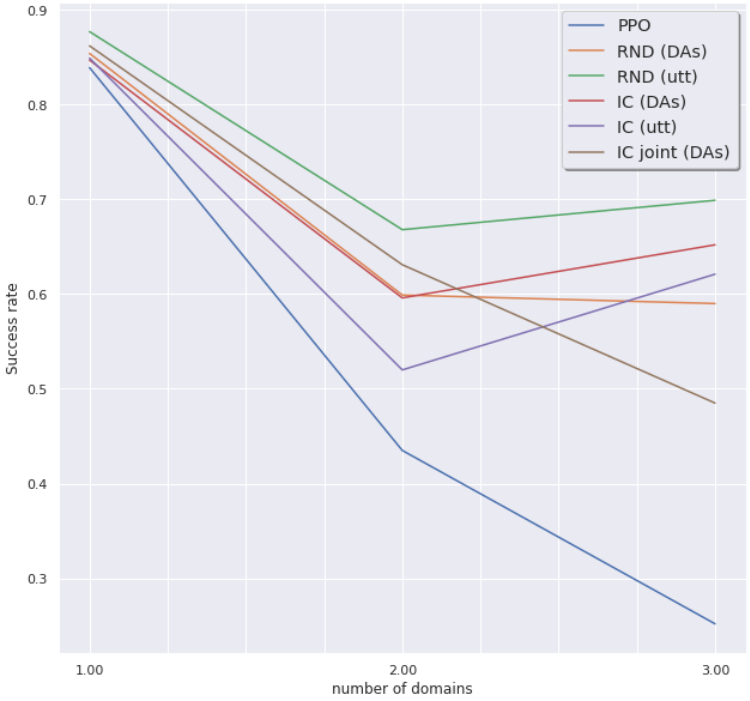

Figure 5.8: Sensitivity of Success rate to the number of domains.

multiple domains and can potentially make it more scalable to bigger environments that include more than three domains.

5.8 Sampled Dialogues for Trained Policies

In this section I show four examples of dialogues between the user simulator and two systems, one using as policy the baseline PPO and the other using RND trained from utterances. For this purpose, four different user goals were sampled as shown in Figure 5.10, covering two single domain goals, one with two domains, and one with three. A simple rule-based NLG module was used to transform the dialogue acts into sentences.

For the first user goal in Example 1, Figure 5.11 shows that PPO is not able to succeed, and fails to provide the requested hotel address[11], while RND (Utt) manages to provide all the information (see Figure 5.12).

Given the second goal in Example 2, both PPO (see Figure 5.13) and RND (Utt) (see Figure 5.14) succeed in the task. However, after the initial user dialogue

[11] This can be seen in the final goal shown in 5.11, where the slot value for the address is "?".

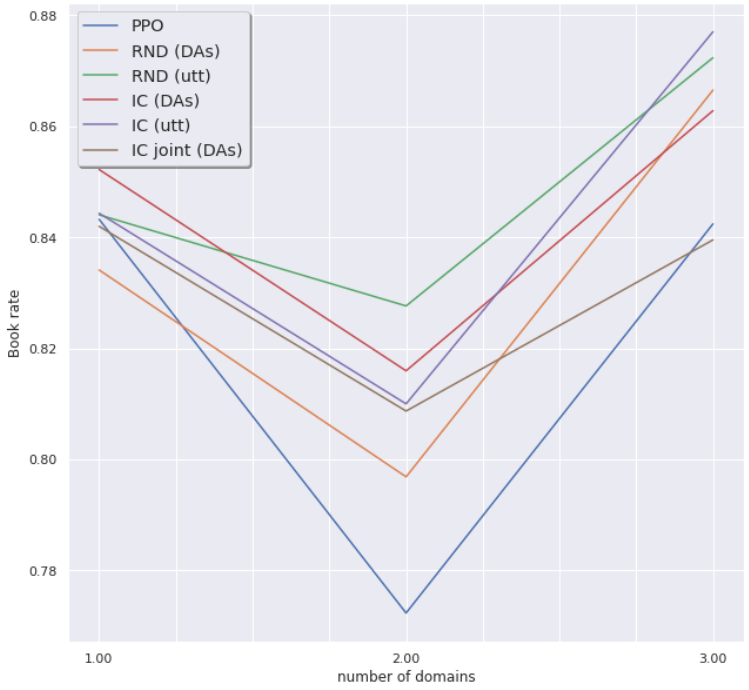

Figure 5.9: Sensitivity of Book rate to the number of domains.

where all the information for the taxi booking is provided, both systems ask again for this same information. This behaviour is due to the simplicity of the NLG module, which makes it to address the user with a set of pre-defined questions. While PPO manages to make the booking, it asks again for the same information, and surprisingly also provides information about a train domain. On the other hand, RND (Utt) makes the booking on its first response, which is a result of the overall lower number of dialogue turns for this model compared to the baseline, as shown in Table 5.2. Nonetheless, both systems finish their dialogues providing again a different taxi booking information, which may be due to the environment not accurately evaluating the belief state of the DST and missing the success on the task, thus providing again the requested information.

Example 3 presents a very simple task where the user is looking for an attraction recommendation, where in particular they want to visit a museum and require to know its phone number. Similar to Example 2, both systems complete the task successfully. However, there is a noticeable difference between how PPO (see Figure 5.15) and RND (Utt) (see Figure 5.16) respond to the user's initial query, where

PPO fails to understand it while RND (Utt) gives a concrete suggestion about a museum as its first response.

Finally, Example 4 shows a more complex initial goal with three domains (attraction, restaurant and taxi), including many information requests per domain and one booking. Without going into the details for both dialogues, it is enough to notice that while both PPO (see Figure 5.17) and RND (Utt) (see Figure 5.18) are successful, this goal is a good example of how limitations of the user simulator can impact in the qualitative evaluation of Dialogue Systems. For example, in the sampled sub-goal for the taxi domain, the user wants a taxi that arrives by a certain time, without knowing (or caring) at what time it should depart, from where and to which destination. In line with the results for taxi shown in 5.3, RND (Utt) manages to correctly decide over the user needs and makes a booking[12] without any intermediate interactions.

[12]For some reason Convlab-2 does not consider that taxis can be booked on its taxi-specific slots, but the DST correctly decides that a taxi of a given brand, and arriving at a certain time, is in fact a booking.

- **Example 1**:

```
{'hotel': {'info': {'area': 'south', 'stars': '4',
                    'type': 'guesthouse'},
           'reqt': {'address': '?'}},
 'train': {'info': {'day': 'thursday',
                    'departure': 'birmingham new street',
                    'destination': 'cambridge',
                    'leaveAt': '21:00'},
           'reqt': {'price': '?', 'trainID': '?'}}}
```

- **Example 2**:

```
{'taxi': {'info': {'arriveBy': '15:00',
                   'departure': 'the fitzwilliam museum',
                   'destination': 'meghna'},
          'reqt': {'car type': '?', 'phone': '?'}}}
```

- **Example 3**:

```
{'attraction': {'info': {'area': 'centre', 'type': 'museum'},
                'reqt': {'phone': '?'}}}
```

- **Example 4**:

```
{'attraction': {'info': {'area': 'centre', 'type': 'museum'},
                'reqt': {'address': '?', 'phone': '?',
                         'postcode': '?'}},
 'restaurant': {'book': {'day': 'wednesday', 'people': '2',
                         'time': '18:45'},
                'booked': '?',
                'fail_book': {'day': 'tuesday', 'people': '2',
                              'time': '18:45'},
                'info': {'area': 'east',
                         'pricerange': 'expensive'},
                'reqt': {'address': '?', 'postcode': '?'}},
 'taxi': {'info': {'arriveBy': '12:15'},
          'reqt': {'car type': '?', 'phone': '?'}}}
```

Figure 5.10: Initial goals for each example

- **User**: I am also looking for information about a train . Can you help me with that ? I need a train going to cambridge . Is it going to birmingham new street ? The earlier the better , so whichever train leaves closest to 21:00.
- **System**: On what day will you be traveling ? The train ID is TR7075 . The fare is 23.60 pounds. Is there anything else I can help you with today ?
- **User**: I would like to leave on thursday.
- **System**: The train ID is TR8272 . The trip will cost 23.60 pounds . The travel time is 51 minutes . Is there anything else I can help you with ?
- **User**: I also need a hotel. I would like a 4 star hotel though , please . Could you help me with a finding a guesthouse ? I ' m also looking for a place to stay . In the south preferably .
- **System**: Is there anything else I can help you with today ?
- **User**: I just need the address .
- **System**: You are welcome . Is there anything else I can help you with today ?
- **User**: Could I get their address ?
- **System**: You are welcome . Is there anything else I can help you with today ?
- **...**
- Final goal:

```
{'hotel': {'info': {'area': 'south', 'stars': '4',
                    'type': 'guesthouse'},
           'reqt': {'address': '?'}},
 'train': {'info': {'day': 'thursday',
                    'departure': 'birmingham new street',
                    'destination': 'cambridge',
                    'leaveAt': '21:00'},
           'reqt': {'price': '23.60 pounds',
                    'trainID': 'TR8272'}}}
```

Figure 5.11: Dialogue example number 1. Model: PPO

- **User**: I am also looking for information about a train . Can you help me with that ? I need a train going to cambridge . Is it going to birmingham new street ? The earlier the better , so whichever train leaves closest to 21:00.
- **System**: On what day will you be traveling ? Where would you like to go to ? What time do you want to depart ?
- **User**: Yes , I need to leave sometimes after 21:00 . Howdy , I need a train heading into cambridge . The train should leave on thursday .
- **System**: The arrival time is 05:51 . Their ID is TR8272 . The price of those tickets are 23.60 pounds .
- **User**: I ' m looking for a hotel . Are there any 4 stars available ? I actually am looking for a guesthouse . I ' m also looking for a place to stay . In the south preferably .
- **System**: The address is 5 mowbray road . It is in the south . There is wifi available at the hotel . The phone number is 01223240089 . The postal code for that hotel is cb17sr . Its listed as moderate . It does have 4 stars . It is a guesthouse . How about aylesbray lodge guest house ? Is there anything else I can help you with ?
- **User**: I am all set . Have a nice day . Bye .
- **System**: The postal code for that hotel is cb17sr . Did you need any further assistance today ?
- Final goal:

```
{'hotel': {'info': {'area': 'south', 'stars': '4',
                    'type': 'guesthouse'},
           'reqt': {'address': '5 mowbray road'}},
 'train': {'info': {'day': 'thursday',
                    'departure': 'birmingham new street',
                    'destination': 'cambridge',
                    'leaveAt': '21:00'},
           'reqt': {'price': '23.60 pounds',
                    'trainID': 'TR8272'}}}
```

Figure 5.12: Dialogue example number 1. Model: RND (utt)

- **User**: I also want to book a taxi to commute between the two places . Taxi to meghna please. I am departing from the fitzwilliam museum . I 'd like to get to the Gallery by 15:00 , please .
- **System**: Where are you departing from ? What is your destination please. What time would you like the taxi to pick you up ?
- **User**: I don't care about the leave of the taxi. I need a taxi to take me to meghna. I ' m at the fitzwilliam museum.
- **System**: There is a train that leaves at 09:00 . Where will you be departing from ? What is your destination please. What time would you like to leave ? The taxi is a honda. The contact number is 54714742922 .
- **User**: Thank you very much , goodbye .
- **System**: I have successfully booked you a toyota . You can reach them on 74597386567. Thank you for using our system !. Is there anything else I can help you with ?
- Final goal:

```
{'taxi': {'info': {'arriveBy': '15:00',
                   'departure': 'the fitzwilliam museum',
                   'destination': 'meghna'},
          'reqt': {'car type': 'honda',
                   'phone': '54714742922'}}}
```

Figure 5.13: Dialogue example number 2. Model: PPO

- **User**: I also want to book a taxi to commute between the two places . Taxi to meghna please. I am departing from the fitzwilliam museum . I 'd like to get to the Gallery by 15:00 , please .
- **System**: What time would you like to arrive there by ? Where will you leave from ? What will your destination be ? What time will you be leaving . A ford is booked . Their contact number is 36395324711.
- **User**: That 's all I need today . Thanks ! Bye !.
- **System**: A honda is booked for you . You can reach them on 71474292285. I ' m happy to help , and I hope you enjoy your stay !. Is there anything else I can help you with ?
- Final goal:

```
{'taxi': {'info': {'arriveBy': '15:00',
                   'departure': 'the fitzwilliam museum',
                   'destination': 'meghna'},
          'reqt': {'car type': 'ford',
                   'phone': '36395324711'}}}
```

Figure 5.14: Dialogue example number 2. Model: RND (utt)

- **User**: I am looking for tourist attractions . How about a museum ? I 'd like a sports place in the centre please .
- **System**: Did you need any further assistance today ?
- **User**: Please get me their phone number.
- **System**: Here is the phone number , 01223314960 . Is there anything else I can help you with ?
- **User**: I am all set . Have a nice day . Bye .
- **System**: Thank you and goodbye .
- Final goal:

```
{'attraction': {'info': {'area': 'centre',
                         'type': 'museum'},
                'reqt': {'phone': '01223314960'}}}
```

Figure 5.15: Dialogue example number 3. Model: PPO

- **User**: I am looking for tourist attractions . How about a museum ? I 'd like a sports place in the centre please .
- **System**: You would love broughton house gallery. It will be located in the centre . We have 11 of those ! Anything specific you need or just a recommendation ? It is a museum . We have broughton house gallery . Is there anything else I can help you with today ?
- **User**: Yes can I get the phone number of one you 'd recommend ?
- **System**: Where did you want to depart from ? The phone number is 01223314960 . Broughton house gallery looks good . Is there anything else I can help you with ?
- **User**: Ok , have a good day . Goodbye .
- **System**: Thank you for using our system !.
- Final goal:

```
{'attraction': {'info': {'area': 'centre',
                         'type': 'museum'},
                'reqt': {'phone': '01223314960'}}}
```

Figure 5.16: Dialogue example number 3. Model: RND (utt)

- **User**: I need a restaurant to dine at in Cambridge on my upcoming trip . I would prefer it in the east area . I ' m looking for an expensive restaurant .
- **System**: Did you have a specific kind of cuisine in mind ? Do you prefer chinese or indian food ? They serve indian food . They serve italian . Booking was successful . Reference number is : 00000001 . Is there anything else I can help you with today ?
- **User**: I don't care about the food of the restaurant.
- **System**: They serve international food . Did you need any further assistance today ?
- **User**: Please give me their address . I would like to know the postcode , if possible ?
- **System**: It is at Finders Corner Newmarket Road. Their postcode is cb259aq. Is there anything else I can help you with today ?
- **User**: Thank you that will be all for now .
- **System**: Did you need any further assistance today ?
- **User**: I ca n't wait to get there and see some of the local attractions . I am looking for a place to stay . Can you help me ? I 'd like a sports place in the centre please . Do you have any museum attractions.
- **System**: Is there anything else I can help you with today ?
- **User**: Just the address please. Can I get the phone number of one ? Can you tell me their postcode ?
- **System**: It is on 98 king street . Here is the phone number , 01223314960 . The post code is cb11ln . Is there anything else I can help you with today ?
- **User**: Now I need a taxi . I want to arrive by 12:15.
- **System**: I need to know where you 'd like picked up at . And where will you be going ? Is there anything else I can help you with ?
- **User**: I don't care about the dest of the taxi. I don't care about the depart of the taxi.
- **System**: What time would you like to be picked up ? The model of the car was bmw . Is there anything else I can help you with ?
- **User**: I don't care about the leave of the taxi.
- **System**: The car arriving will be a bmw . You can give them a call at 79613662456. Did you need any further assistance today ?
- **User**: Thank you , goodbye.
- **System**: Goodbye . If you think of anything else you need do n't hesitate to contact us . Is there anything else I can help you with today ?
- Final goal:

```
{'attraction': {'info': {'area': 'centre',
                         'type': 'museum'},
                'reqt': {'address': '98 king street',
                         'phone': '01223314960',
                         'postcode': 'cb11ln'}},
 'restaurant': {'book': {'day': 'wednesday',
                         'people': '2', 'time': '18:45'},
                'booked': 'yes',
                'fail_book': {'day': 'tuesday',
                              'people': '2', 'time': '18:45'},
                'info': {'area': 'east',
                         'pricerange': 'expensive'},
                'reqt': {'address': 'Finders Corner Newmarket Road',
                         'postcode': 'cb259aq'}},
 'taxi': {'info': {'arriveBy': '12:15'},
          'reqt': {'car type': 'bmw',
                   'phone': '79613662456'}}}
```

Figure 5.17: Dialogue example number 4. Model: PPO

- **User**: I need a restaurant to dine at in Cambridge on my upcoming trip . I would prefer it in the east area . I ' m looking for an expensive restaurant .
- **System**: Did you have a specific kind of cuisine in mind ?
- **User**: I don't care about the food of the restaurant.
- **System**: (no dialogue)
- **User**: Thank you . May I have the address ? Can I get the postcode for it please ?
- **System**: They are located at Finders Corner Newmarket Road. Their postcode is cb259aq. You are welcome . Is there anything else I can help you with today ?
- **User**: I would like to book a table for tuesday. Yes . How about 18:45 ? It will be for 2 people .
- **System**: Your reference number is 00000001 . Did you need any further assistance today ?
- **User**: Great that 's all that I needed to know , thank you !.
- **System**: You are welcome . Is there anything else I can help you with today ?
- **User**: Can you help me plan a trip to see a particular attraction ? I am also looking for suggestions on places to go in the centre of town . Can you help me with that ? How about a museum ?
- **System**: How about broughton house gallery ? they 're pretty fun . It 's located in the centre . Sure , there are 11 options for you. It is a museum . You are welcome . Is there anything else I can help you with today ?
- **User**: That sounds great ! May I have the address ? Can you give me their phone number please ? Can you tell me their postcode ?
- **System**: It is on 6 trinity street . How about broughton house gallery ? they 're pretty fun . Here is the phone number , 01223314960 . Its postcode is cb11ln . 98 king street is the address. Their entrance fee is free by our system currently . Here is the phone number , 01223314960 . Its postcode is cb11ln . Broughton house gallery looks good . Is there anything else I can help you with today ?
- **User**: Now I need a taxi . I 'd like the arrival time to be 12:15 , please .
- **System**: The taxi is due to arrive at none . A lexus is booked . I have booked a taxi to pick you up at none . I have booked a taxi to take you to none for your reservation . It will leave at none . You can reach them on 23573393958. You are welcome . Is there anything else I can help you with today ?
- **User**: Thank you , goodbye.
- **System**: I ' m happy to help , and I hope you enjoy your stay !.
- Final goal:

```
{'attraction': {'info': {'area': 'centre', 'type': 'museum'},
                'reqt': {'address': '6 trinity street',
                         'phone': '01223314960',
                         'postcode': 'cb11ln'}},
 'restaurant': {'book': {'day': 'wednesday', 'people': '2',
                         'time': '18:45'},
                'booked': 'yes',
                'fail_book': {'day': 'tuesday', 'people': '2',
                              'time': '18:45'},
                'info': {'area': 'east',
                         'pricerange': 'expensive'},
                'reqt': {'address': 'Finders Corner Newmarket Road',
                         'postcode': 'cb259aq'}},
 'taxi': {'info': {'arriveBy': '12:15'},
          'reqt': {'car type': 'lexus',
                   'phone': '23573393958'}}}
```

Figure 5.18: Dialogue example number 4. Model: RND (Utt)

Chapter 6

Conclusions and Future Work

In this work I explored intrinsic motivation models to improve exploration in Dialogue Systems. I used the Convlab-2 environment with MultiWOZ, which is a standard multi-domain task-oriented DS dataset. In particular, I implemented Random Network Distillation and Intrinsic Curiosity methods, that are both based on the dialogue acts and semantic utterances between the agent and the system. I have found that these approaches helped to increase the performance of the system policy. The best result was achieved for the case when intrinsic rewards were generated through RND based on utterances, with a success rate of 73 percent, improving over the baseline PPO, which has a success rate of 60 percent. Moreover, complete and book rates were increased by 10 percent with respect to the baseline for this utterance-based RND approach. I also showed that these intrinsic motivation models make the system policy more robust to an increasing number of domains, suggesting that they may be a good approach for scaling to environments containing an even bigger number of domains.

While using the semantic similarity between agent and system utterances was shown to yield good results, there are limitations due to the simplicity of the DS modules that were used in this work. In particular,

- The NLG module can be improved by using a language model-based approach instead of a rule-based one. This will lead to better quality and more realistic utterances;
- The DST module is also critical when training system policies. Rule-based

DST does not take into account that the dataset is multi-domain, therefore it takes one domain at a time while interacting with the user. Employing more sophisticated DST algorithms rather than the rule-based approach may improve the overall DS performance;

- The User simulator should be improved so that it tries to achieve more human-like goals, compared to the current rule-based one.

Moreover, one of the main limitations of this work was using a very recently released library as in the form of Convlab-2. During the course of this project there were many different updates and modifications to Convlab-2, therefore it became necessary to work with a stable version ignoring any posterior changes. While some of the bugs and incorrect implementations were already addressed since release, some others remain. In particular:

- The user simulator is very simple and generates a distribution of user goals that are not realistic enough to produce real world-like dialogues. For example, one shown user goal was asking for a taxi without specifying either the departure or arrival locations or the arriving time. Improvements on Convlab-2's Policy Agenda model for the user simulator will be needed;

- Agent and environment methods showed many inconsistencies when computing performance metrics or simply indicating if the dialogue was successful or not. The latter had a direct impact on training and evaluation and had to be corrected.

- Environment rewards, as based on Convlab-2's task success method, were also broken. For example, there were cases where the environment yielded a constant stream of positive rewards before the dialogue was finished, due to the success on one single domain. Again, this was corrected during the training algorithm by defining a step-wise reward of $+40$ if the dialogue was successful for all the domains, and -1 otherwise.

There are many possible avenues for future work, both in the general field of studying Intrinsic Motivation for Reinforcement Learning of Dialogue System Policies,

as well as for carrying out research using the Convlab-2 library in particular. To begin with, all the experiments can be replicated using more sophisticated NLG, DST and user simulator models. For the latter, if there are no improvements in Convlab-2, a better user simulator should be implemented independently. It would also be interesting to extend this work to environments containing different datasets, for example those included in Convlab-2 apart from MultiWOZ. On the other hand, the intrinsic motivation models investigated in this work could be improved by taking into account the *length* of each dialogue instance, for example generating an intrinsic reward signal not only transition-based (as in RND and IC) but also episode-based (as in [47]), where there could be an extra reward bonus for handling different domains at once. Finally, combining intrinsic motivation models with different reward shaping strategies could help to further improve the reward signal seen by the agent at every environment step, which can potentially lead to an increasing policy performance.

Bibliography

[1] ASRI, L. E., HE, J., AND SULEMAN, K. A sequence-to-sequence model for user simulation in spoken dialogue systems. *CoRR abs/1607.00070* (2016).

[2] AUBRET, A., MATIGNON, L., AND HASSAS, S. A survey on intrinsic motivation in reinforcement learning, 2019.

[3] BADIA, A. P., PIOT, B., KAPTUROWSKI, S., SPRECHMANN, P., VITVITSKYI, A., GUO, D., AND BLUNDELL, C. Agent57: Outperforming the atari human benchmark, 2020.

[4] BARTO, A. G. Intrinsic motivation and reinforcement learning. *Intrinsically Motivated Learning in Natural and Artificial Systems 1* (2013), 14–47.

[5] BELLEMARE, M. G., NADDAF, Y., VENESS, J., AND BOWLING, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research 47* (Jun 2013), 253–279.

[6] BELLEMARE, M. G., SRINIVASAN, S., OSTROVSKI, G., SCHAUL, T., SAXTON, D., AND MUNOS, R. Unifying count-based exploration and intrinsic motivation, 2016.

[7] BROCKMAN, G., CHEUNG, V., PETTERSSON, L., SCHNEIDER, J., SCHULMAN, J., TANG, J., AND ZAREMBA, W. Openai gym, 2016.

[8] BUDZIANOWSKI, P., WEN, T.-H., TSENG, B.-H., CASANUEVA, I., ULTES, S., RAMADAN, O., AND GAŠIĆ, M. 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, Oct.-Nov. 2018), Association for Computational Linguistics, pp. 5016–5026.

[9] BUDZIANOWSKI, P., WEN, T.-H., TSENG, B.-H., CASANUEVA, I., ULTES, S., RAMADAN, O., AND GAŠIĆ, M. Multiwoz – a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling, 2018.

[10] BURDA, Y., EDWARDS, H., STORKEY, A., AND KLIMOV, O. Exploration by random network distillation, 2018.

[11] CHANDRAMOHAN, S., GEIST, M., LEFÈVRE, F., AND PIETQUIN, O. User Simulation in Dialogue Systems using Inverse Reinforcement Learning. In *Interspeech 2011* (Florence, Italy, Aug. 2011), pp. 1025–1028.

[12] DEVLIN, J., CHANG, M., LEE, K., AND TOUTANOVA, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR abs/1810.04805* (2018).

[13] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[14] GAO, J., GALLEY, M., AND LI, L. Neural approaches to conversational ai, 2018.

[15] GATT, A., AND KRAHMER, E. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation, 2017.

[16] GORDON-HALL, G., GORINSKI, P. J., AND COHEN, S. B. Learning dialog policies from weak demonstrations, 2020.

[17] GORDON-HALL, G., GORINSKI, P. J., LAMPOURAS, G., AND IACOBACCI, I. Show us the way: Learning to manage dialog from demonstrations, 2020.

[18] GUR, I., HAKKANI-TUR, D., TUR, G., AND SHAH, P. User modeling for task oriented dialogues, 2018.

[19] HECK, M., VAN NIEKERK, C., LUBIS, N., GEISHAUSER, C., LIN, H.-C., MORESI, M., AND GAŠIĆ, M. Trippy: A triple copy strategy for value independent neural dialog state tracking, 2020.

[20] HENDERSON, M., THOMSON, B., AND WILLIAMS, J. D. The second dialog state tracking challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)* (Philadelphia, PA, U.S.A., June 2014), Association for Computational Linguistics, pp. 263–272.

[21] HESTER, T., VECERIK, M., PIETQUIN, O., LANCTOT, M., SCHAUL, T., PIOT, B., HORGAN, D., QUAN, J., SENDONARIS, A., DULAC-ARNOLD, G., OSBAND, I., AGAPIOU, J., LEIBO, J. Z., AND GRUSLYS, A. Deep q-learning from demonstrations, 2017.

[22] KEIZER, S., GASIC, M., JURCÍCEK, F., MAIRESSE, F., THOMSON, B., YU, K., AND YOUNG, S. Parameter estimation for agenda-based user simulation. pp. 116–123.

[23] KONDA, V., AND TSITSIKLIS, J. Actor-critic algorithms. *Society for Industrial and Applied Mathematics 42* (04 2001).

[24] KREYSSIG, F., CASANUEVA, I., BUDZIANOWSKI, P., AND GAŠIĆ, M. Neural user simulation for corpus-based policy optimisation of spoken dialogue systems. pp. 60–69.

[25] LEE, S., ZHU, Q., TAKANOBU, R., LI, X., ZHANG, Y., ZHANG, Z., LI, J., PENG, B., LI, X., HUANG, M., AND GAO, J. Convlab: Multi-domain end-to-end dialog system platform. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019).

[26] LEI, W., JIN, X., KAN, M.-Y., REN, Z., HE, X., AND YIN, D. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. pp. 1437–1447.

[27] LEWIS, M., YARATS, D., DAUPHIN, Y. N., PARIKH, D., AND BATRA, D. Deal or no deal? end-to-end learning for negotiation dialogues, 2017.

[28] LI, J., MONROE, W., RITTER, A., GALLEY, M., GAO, J., AND JURAFSKY, D. Deep reinforcement learning for dialogue generation, 2016.

[29] LI, L., WILLIAMS, J., AND BALAKRISHNAN, S. Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection. pp. 2475–2478.

[30] LIPTON, Z. C., LI, X., GAO, J., LI, L., AHMED, F., AND DENG, L. Bbq-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems, 2016.

[31] LOSHCHILOV, I., AND HUTTER, F. Decoupled weight decay regularization, 2017.

[32] MAIRESSE, F., GASIC, M., JURCÍCEK, F., KEIZER, S., THOMSON, B., YU, K., AND YOUNG, S. Spoken language understanding from unaligned data using discriminative classification models. pp. 4749–4752.

[33] MAIRESSE, F., AND YOUNG, S. Stochastic language generation in dialogue using factored language models. *Computational Linguistics 40*, 4 (Dec. 2014), 763–799.

[34] MEHRI, S., SRINIVASAN, T., AND ESKENAZI, M. Structured fusion networks for dialog, 2019.

[35] MESNIL, G., DAUPHIN, Y., YAO, K., BENGIO, Y., DENG, L., HAKKANI-TUR, D., HE, X., HECK, L., TUR, G., YU, D., AND ZWEIG, G. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing 23*, 3 (2015), 530–539.

[36] MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., AND RIEDMILLER, M. Playing atari with deep reinforcement learning, 2013.

[37] MRKŠIĆ, N., Ó SÉAGHDHA, D., WEN, T.-H., THOMSON, B., AND YOUNG, S. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Vancouver, Canada, July 2017), Association for Computational Linguistics, pp. 1777–1788.

[38] OSTROVSKI, G., BELLEMARE, M. G., VAN DEN OORD, A., AND MUNOS, R. Count-based exploration with neural density models, 2017.

[39] OUDEYER, P., KAPLAN, F., AND HAFNER, V. V. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation 11*, 2 (2007), 265–286.

[40] PAPINENI, K., ROUKOS, S., WARD, T., AND ZHU, W. J. Bleu: a method for automatic evaluation of machine translation.

[41] PATHAK, D., AGRAWAL, P., EFROS, A. A., AND DARRELL, T. Curiosity-driven exploration by self-supervised prediction, 2017.

[42] PIETQUIN, O., GEIST, M., CHANDRAMOHAN, S., AND FREZZA-BUET, H. Sample-efficient batch reinforcement learning for dialogue management optimization. *ACM Trans. Speech Lang. Process. 7*, 3 (June 2011).

[43] PIETQUIN, O., AND HASTIE, H. A survey on metrics for the evaluation of user simulations. *The Knowledge Engineering Review 28*, 1 (2013), 59–73.

[44] RAVURI, S., AND STOLCKE, A. Recurrent neural network and lstm models for lexical utterance classification. *International Speech Communication Association* (10 2015), 135–.

[45] ROSS, S., GORDON, G. J., AND BAGNELL, J. A. A reduction of imitation learning and structured prediction to no-regret online learning, 2010.

[46] RUDER, S. An overview of gradient descent optimization algorithms, 2016.

[47] SAVINOV, N., RAICHUK, A., MARINIER, R., VINCENT, D., POLLEFEYS, M., LILLICRAP, T., AND GELLY, S. Episodic curiosity through reachability, 2018.

[48] SCHATZMANN, J., THOMSON, B., WEILHAMMER, K., YE, H., AND YOUNG, S. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers* (Rochester, New York, Apr. 2007), Association for Computational Linguistics, pp. 149–152.

[49] SCHATZMANN, J., WEILHAMMER, K., STUTTLE, M., AND YOUNG, S. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowl. Eng. Rev. 21*, 2 (June 2006), 97–126.

[50] SCHMIDHUBER, J. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development 2*, 3 (2010), 230–247.

[51] SCHULMAN, J., LEVINE, S., MORITZ, P., JORDAN, M. I., AND ABBEEL, P. Trust region policy optimization, 2015.

[52] SCHULMAN, J., MORITZ, P., LEVINE, S., JORDAN, M., AND ABBEEL, P. High-dimensional continuous control using generalized advantage estimation, 2015.

[53] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A., AND KLIMOV, O. Proximal policy optimization algorithms, 2017.

[54] SHI, W., QIAN, K., WANG, X., AND YU, Z. How to build user simulators to train rl-based dialog systems, 2019.

[55] SINGH, S., KEARNS, M., LITMAN, D., AND WALKER, M. Reinforcement learning for spoken dialogue systems.

[56] SU, P.-H., VANDYKE, D., GASIC, M., KIM, D., MRKSIC, N., WEN, T.-H., AND YOUNG, S. Learning from real users: Rating dialogue success with neural networks for reinforcement learning in spoken dialogue systems, 2015.

[57] SUTTON, R. S., AND BARTO, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[58] SZEPESVARI, C. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[59] TAKANOBU, R., ZHU, H., AND HUANG, M. Guided dialog policy learning: Reward estimation for multi-domain task-oriented dialog, 2019.

[60] TANG, H., HOUTHOOFT, R., FOOTE, D., STOOKE, A., CHEN, X., DUAN, Y., SCHULMAN, J., TURCK, F. D., AND ABBEEL, P. exploration: A study of count-based exploration for deep reinforcement learning, 2016.

[61] ULTES, S., BUDZIANOWSKI, P., CASANUEVA, I., MRKŠIĆ, N., ROJAS-BARAHONA, L., SU, P.-H., WEN, T. H., GAŠIĆ, M., AND YOUNG, S. Domain-independent user satisfaction reward estimation for dialogue policy learning. pp. 1721–1725.

[62] VINYALS, O., AND LE, Q. A neural conversational model, 2015.

[63] WANG, Z., BAPST, V., HEESS, N., MNIH, V., MUNOS, R., KAVUKCUOGLU, K., AND DE FREITAS, N. Sample efficient actor-critic with experience replay, 2016.

[64] WATKINS, C. J., AND DAYAN, P. Q-learning, 1992.

[65] WEI, X., AND RUDNICKY, A. Task-based dialog management using an agenda.

[66] WEN, T.-H., GAŠIĆ, M., KIM, D., MRKŠIĆ, N., SU, P.-H., VANDYKE, D., AND YOUNG, S. Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue* (Prague, Czech Republic, Sept. 2015), Association for Computational Linguistics, pp. 275–284.

[67] WEN, T.-H., GASIC, M., MRKSIC, N., SU, P.-H., VANDYKE, D., AND YOUNG, S. Semantically conditioned lstm-based natural language generation for spoken dialogue systems, 2015.

[68] WEN, T.-H., GAŠIĆ, M., MRKŠIĆ, N., SU, P.-H., VANDYKE, D., AND YOUNG, S. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (Lisbon, Portugal, Sept. 2015), Association for Computational Linguistics, pp. 1711–1721.

[69] WESSELMANN, P., WU, Y., AND GAŠIĆ, M. Curiosity-driven reinforcement learning for dialogue management. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2019), pp. 7210–7214.

[70] WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn. 8*, 3–4 (May 1992), 229–256.

[71] WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8* (1992), 229–256.

[72] WU, C.-S., MADOTTO, A., HOSSEINI-ASL, E., XIONG, C., SOCHER, R., AND FUNG, P. Transferable multi-domain state generator for task-oriented dialogue systems, 2019.

[73] XU, P., AND HU, Q. An end-to-end approach for handling unknown slot values in dialogue state tracking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Melbourne, Australia, July 2018), Association for Computational Linguistics, pp. 1448–1457.

[74] ZHANG, Z., TAKANOBU, R., ZHU, Q., HUANG, M., AND ZHU, X. Recent advances and challenges in task-oriented dialog system, 2020.

[75] ZHONG, V., XIONG, C., AND SOCHER, R. Global-locally self-attentive encoder for dialogue state tracking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Melbourne, Australia, July 2018), Association for Computational Linguistics, pp. 1458–1467.

[76] ZHU, Q., HUANG, K., ZHANG, Z., ZHU, X., AND HUANG, M. Crosswoz: A large-scale chinese cross-domain task-oriented dialogue dataset. *Transactions of the Association for Computational Linguistics 8* (2020), 281–295.

[77] ZHU, Q., ZHANG, Z., FANG, Y., LI, X., TAKANOBU, R., LI, J., PENG, B., GAO, J., ZHU, X., AND HUANG, M. Convlab-2: An open-source toolkit for building, evaluating, and diagnosing dialogue systems, 2020.

[78] ZHU, Q., ZHANG, Z., FANG, Y., LI, X., TAKANOBU, R., LI, J., PENG, B., GAO, J., ZHU, X., AND HUANG, M. Convlab-2: An open-source toolkit for building, evaluating, and diagnosing dialogue systems, 2020.