An Adaptive Dialogue System with Online Dialogue Policy Learning

Alexandros Papangelis[1,2], Nikolaos Kouroupas[3], Vangelis Karkaletsis[1], and Fillia Makedon[2]

[1] National Centre for Scientific Research "Demokritos", Institute of Informatics and Telecommunications
[2] University of Texas at Arlington Department of Computer Science and Engineering
[3] University of Piraeus, Department of Informatics
alexandros.papangelis@mavs.uta.edu, vangelis@iit.demokritos.gr, makedon@uta.edu, nick_kouroupas@hotmail.com

Abstract. In this work we present an architecture for Adaptive Dialogue Systems and a novel system that serves as a Museum Guide. It employs several online Reinforcement Learning (RL) techniques to achieve adaptation to the environment as well as to different users. Not many systems have been proposed that apply online RL methods and this is one of the first to fully describe an Adaptive Dialogue System with online dialogue policy learning. We evaluate our system through user simulations and compare the several implemented algorithms on a simple scenario.

Keywords: Adaptive Dialogue Systems, Reinforcement Learning.

1 Introduction

Dialogue Systems (DS) are systems that are able to make human-like conversation with their users and are widely used, mainly as service providers (customer support, flight or hotel booking and others). Such systems, however, are static and inflexible to individual user needs or to changes in their environment, giving rise to a growing need for adaptation in DS. Adaptive Dialogue Systems (ADS) are rapidly becoming smarter and more complex as novel or state of the art Artificial Intelligence (AI) techniques are being applied. AI is a very important part of ADS as it provides methods for learning optimal dialogue policies, Natural Language Understanding (NLU), Natural Language Generation (NLG) strategies and more. In this work we propose an ADS architecture and a system that automatically learns optimal dialogue policies using online RL algorithms.

To the best of our knowledge there is a limited amount of works on dialogue policy learning and due to space constraints we will only mention a few. Cuayáhuitl et al. [2] propose a travel planning DS, that uses hierarchical Reinforcement Learning (RL) to learn dialogue policies for complex actions.

I. Maglogiannis, V. Plagianakos, and I. Vlahavas (Eds.): SETN 2012, LNAI 7297, pp. 323–330, 2012.

Young et al. [14] propose an ADS that provides venue information to tourists in a fictitious city that is able to handle uncertainty and misunderstandings. These systems, however, learn dialogue policies in an offline fashion, meaning that when they interact with real users they actually follow an optimal but static policy. Pietquin et al. [7] propose a system that provides information about restaurants and follow a particle filtering approach to dialogue policy learning. The authors propose the Kalman Temporal Differences algorithm, that is able to learn policies online while also being sample-efficient compared to other methods. Gašić et al. [3] propose a system that provides information about restaurants in Cambridge and applies online RL using Gaussian Processes to learn dialogue policies. These approaches, however, while having many advantages such as handling uncertainty or error recovery, are expensive to run since in [7] many dialogues are required to solve a small problem and in [3] the improvement was not statistically significant over a handcrafted policy.

Our main contribution is that we propose an architecture for ADS that applies several online RL algorithms for dialogue policy learning that are simple to implement and of low cost to run. Such a system is able to learn policies from simulated users for several user categories and these policies are then used as initial policies when the system interacts with a real user. One can also provide handcrafted policies when expert knowledge is available. The system is therefore able to rapidly adapt to a new user since it exploits prior knowledge and continues to learn throughout the interaction.

In the next section we will briefly present some necessary background knowledge, in section 3 we will review the design of our system, in section 4 we will present our evaluation and in section 5 we will discuss the results and conclude.

2 Background

In this section we will briefly provide some details on Markov Decision Processes (MDP), which is the model used by RL, as well as details about RL itself to better understand how it is applied to learn optimal dialogue policies in our system.

A MDP is a tuple $M = \{S, A, P\}$, composed of a non empty set of states S, a non empty set of actions A and a transition probability kernel P that, for each triplet (s_1, a, s_2), models the probability of moving from state s_1 to state s_2 when taking action a. When the system transits from a state to another it receives an immediate feedback value r, called reward. We define a reward function to be the expected value of that reward: $R(s, a) = \mathbb{E}[r(s, a)]$. Here $r(s, a)$ represents the immediate reward the system received after taking action a [11]. A policy π dictates the behaviour of a system and is defined as a mapping from $S \times A$ to a distribution $\pi(s, \alpha)$ that models the probability according to which the system will take action α when it is in state s. A sequence of states and actions until a terminal state is reached defines an episode. For example, a search robot modelled as an MDP may be in a state reflecting its location, and the available actions may be the directions it can go. It will receive positive

feedback for finding what it was looking for and negative analogous to the time spent searching. A policy would describe a way of searching the space and RL would try, through the robot's interaction with the environment, to find an optimal way of searching.

More formally, RL tries to find (or learn) an optimal policy π for a system modelled as an MDP. To this aim a function $V(s)$, called *value function*, must be defined that will provide an estimate of how good a policy is $V^\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s]$, $s \in S$, i.e. $V^\pi(s)$ yields the expected cumulative discounted rewards when the system begins from state s and follows policy π. The discounting factor $\gamma \in [0, 1]$ captures the significance of future rewards. The *return* of a policy π is defined as: $J^\pi = \sum_{t=0}^{\infty} \gamma^t r_t(s_t, \pi(s_t))$, where $\pi(s_t) = argmax_\alpha\{\pi(s_t, \alpha)\}$. For an optimal policy $\pi^\star$ we have that $J^{\pi^\star}(s) = V^{\pi^\star}(s), \forall s \in S$. The *action-value function* is defined as $Q^\pi(s, \alpha) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = \alpha]$, where $s \in S, \alpha \in A$. Function $Q^\pi(s, a)$ provides the expected cumulative discounted rewards when the system begins from state s and takes action α, following policy π. [11]

RL algorithms can be categorised into model-free and model-based algorithms. The first, as the name implies, do not use any model of the environment to aid the learning process, while the latter use interactions with the environment as well as interactions with a model that simulates the world to speed up learning. To implement model-based algorithms we used the Dyna [10] architecture.

3 System Overview

Our system is designed to act as a museum guide and provide descriptions of exhibits in a virtual museum. It is based on the INDIGO [5] system and was implemented using the Olympus [1] platform, which is a platform for developing dialogue systems for research purposes. INDIGO is an affective museum guide dialogue system, able to adapt to different user personalities and user expertise levels. It can assess the user's mood and emotional state and adapt its output accordingly. For dialogue management we used Olympus' RavenClaw Dialogue Manager (DM) and extended it with online RL modules.

Figure 1 depicts the architecture of the proposed Museum Guide dialogue system, where s, a, r are the current state, previous action and reward received respectively and s', a' are the new state and new action. $\tilde{u}_u$ and $\tilde{u}_s$ are the noisy user and system utterances while u_u and u_s are the interpreted user utterance and actual system utterance respectively. The system uses Olympus' NLU and NLG components and also has a learning component, where all learning algorithms are implemented. The DM receives u_u which contains the reward of the last action r. It then sends s, a and r to the learning component and receives back the new system state s' and the new action a' it should take. It then takes action a' and sends a description of the system's utterance u_s to the NLG component. Last, it has an ontology that is based on INDIGO's ontology, which describes museum artefacts, time periods, persons and more and contains information about many exhibits. Museum Guide currently supports one type of

query and that is requesting for a description of an exhibit. Search in the ontology is performed by providing values for the exhibit's type, construction time, or time period if it is a person and its physical location.

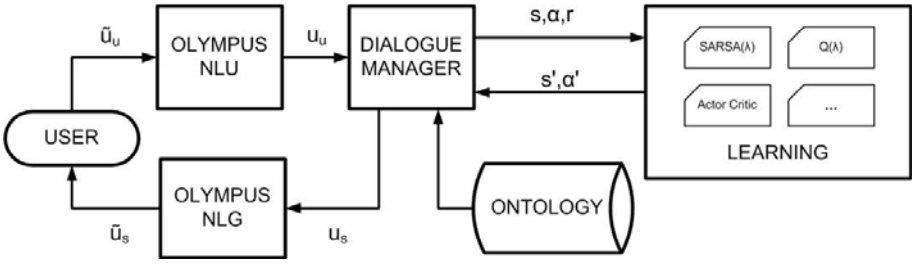

Fig. 1. Architecture of the proposed Museum Guide System

To achieve adaptation in ADS researchers should carefully select methods that are able to tackle the many challenges of this field. RL can successfully handle many of those, such as error recovery and robustness to environmental changes. More specifically, RL can be applied in Dialogue Management (DM) in order to find an optimal dialogue policy that will yield the best action the system should take, depending on the state it is in. Using online RL techniques, the system is able to learn continuously and adapt to changes or different users.

We have implemented a variety of online RL algorithms covering a broad range of the available methods. Our system therefore is able to continue learning as it interacts with real users as well as switch learning methods at will, depending on the problem at hand. Each algorithm's output is in a standardized form and so policies are interchangeable, meaning we can learn a policy using one algorithm and apply it using another. This gives the designer the option of providing a handcrafted policy (modeling prior knowledge) to the system and the system will then optimize that policy according to its current needs. One can also have several user categories and provide handcrafted policies or policies learned through simulations, for each category. The system will be able to use them as initial policies when interacting with appropriate users and refine them to adapt to the specific users' needs. We will now briefly describe the algorithms implemented in our system.

SARSA(λ) (S(λ)) is a very popular RL algorithm, often used in ADS. It applies temporal difference methods for learning and produces an estimate of $Q^\pi(s, a)$, represented as a matrix [10]. λ is a parameter thar controls how much effect will past experience have in future updates. SARSA(λ) is a model-free algorithm, but we also have a model-based version implemented, called DynaSARSA(λ) (DS(λ)).

Q-Learning (Q) was proposed by Watkins [12] and it also learns an estimate of $Q^\pi(s, a)$. The major differences with SARSA(λ) are that it calculates the difference between $Q^\pi(s', a^*)$ and $Q^\pi(s, a)$, where s' is the new state and a^* is

the greedily optimal action, that it only updates a single entry of the matrix representing $Q^\pi(s,a)$ at each iteration and does not use eligibility traces, which is a matrix that keeps track of states visited and actions taken. We also have a Dyna version implemented for this algorithm (DQ).

Q(λ) is similar to Q-Learning in the sense that it uses the temporal difference between $Q^\pi(s',a^*)$ and $Q^\pi(s,a)$ and similar to SARSA(λ) in the sense that it updates the whole $Q^\pi(s,a)$ matrix at each iteration and uses eligibility traces [12,6]. There is a Dyna version for Q(λ) as well (DQ(λ)).

Actor Critic QV (AC) is an algorithm that uses an actor to follow the policy that is being learned and a critic to evaluate it. This algorithm uses the temporal difference between $Q^\pi(s,a)$ and $V^\pi(s)$ to compute updates for the actor and the critic [13]. We have also implemented a model-based version for this algorithm (DAC).

Our system can operate in two modes. Either in *simulation mode*, where the learning algorithms are trained using simulated users, or in *real user mode* where the system interacts with real users while continuing to learn at a lower pace. A policy learnt using either mode can be saved and loaded for later use again in either mode. During training the system interacts with a real or simulated user and receives a reward after each action (i.e. system utterance). Through this process the system learns which actions bring higher rewards and in what order they should be taken. At the initial stages of training, RL algorithms explore the state-action space and, therefore, make many mistakes. Below we can see example dialogues before training, where actions are taken in a near random way and after training with SARSA(λ):

Dialogue 1: Untrained System
S: Welcome to Museum Guide!
S: What is the location you are looking for?
U: In Athens
S: What is it you'd like to learn about?
U: An altar
S: What is the time period you are looking for?
U: Around 100bc
S: What is the time period you are looking for?
U: Around 100bc
S: What is the location you are looking for?
U: In Athens
S: What is the time period you are looking for?
U: Around 100bc
S: What is the location you are looking for?
U: In Athens
S: Welcome to Museum Guide!
⋮ (omitted)
S: [ITEM DESCRIPTION]
S: You can say, start a new query, or goodbye.
U: bye!
S: Thank you for using Museum Guide, goodbye!

Dialogue 2: Trained System
S: Welcome to Museum Guide!
S: What is it you'd like to learn about?
U: An altar
S: What is the time period you are looking for?
U: Around 100bc
S: What is the location you are looking for?
U: In Athens
S: [ITEM DESCRIPTION]
S: You can say, start a new query, or goodbye.
U: bye!
S: Thank you for using Museum Guide, goodbye!

4 Evaluation

Evaluation of ADS or DS in general is still an open question. Many approaches and methods have been proposed as listed in detail in [8]. When researchers use RL methods the most common approach is to use the reward function $R(s, a)$ as a metric of performance, since it is explicitly defined to minimize or maximize standard metrics such as dialogue length or goal achievement. In our experiments we evaluated each algorithm's speed of convergence, using the reward function $R(d, a)$ as defined later in equation (1). In the rest of this section we will formally model the dialogue problem, describe our experimental setup and present the results of our evaluation, where we focused on the comparison of the different techniques. For simplicity, we opted for a noise free scenario.

The dialogue problem can be formulated as a *slot filling problem*. In such a setting, for example, we have a system that needs to retrieve an exhibit from a database and present it to the user. The user probably has in mind something like: "I want to learn about an *altar*, dating from *500 BC* and located near *Athens*." The information that the system needs (which is item attribute values in its database) are in *italics*, and are called slots. The system must then ask a series of questions in order to retrieve the required information i.e. fill those slots. For different queries the system will require different slots to be filled.

More formally, we can define the problem as: $Z =< z_0, ..., z_N >\in M, M = M_0 \times M_1 \times ... \times M_N, M_i = \{1, ..., |M_i|\}$, where Z are the N slots the user needs to fill and each slot z_i belongs in the set M_i. We can also define the dialogue state as a vector $d \in M$, where the dimensions correspond to slots and their values correspond to the slot values. We also define system actions $A \in \{1, ..., |Z|\}$ to be requests for slots and specifically a_i will request slot z_i. It is possible to define a set of available actions $\tilde{a}_i \subset A$ at each dialogue state d_i but we let $\tilde{a}_i = A$ for simplicity. The user goals are represented by a query vector $q \subset Z$, that represents the slots needed to be filled in order for the system to accurately provide an answer. Please note that we set action a_N to mean *Give Answer*. The reward function in our system is defined as:

$$R(d, a) = \begin{cases} -1, & \text{if} \quad a \neq a_N \\ -100, & \text{if} \quad a = a_N, \exists q_i | q_i = \emptyset \\ 0, & \text{if} \quad a = a_N, \neg\exists q_i | q_i = \emptyset \end{cases} \tag{1}$$

Thus, the optimal reward for each problem is: $-|q|$ since $|q| < |Z|$. Note that this reward function penalises long dialogues and inaccurate responses (i.e. attempts to answer without enough information). It is straightforward to extend this model to account for uncertainty but we will not discuss this in this work.

In our evaluation our problem had 3 slots, *Type, TimePeriod* and *Location*, and typically 6 actions, *Welcome, AskType, AskTimePeriod, AskLocation, GreetGoodbye* and *GiveAnswer*, out of which we hardcoded the first action to always be *Welcome* and the last to be *GreetGoodbye*. The system then needed to learn how to retrieve the three slots in the most efficient way. We evaluated the system using a simple noise-free user simulator that always responds correctly to any

system request. An episode in this problem is over when the system presents the results to the user (which may or may not be the right thing to do). For each algorithm we counted the number of episodes it took to learn the optimal policy and averaged it over 25 runs. Note here that convergence speed is correlated with the average total reward, since the sooner the algorithm converges the higher the reward (it will be performing optimally from the point of convergence and after). The results are shown in Table 1, below.

Table 1. Average learning speed

Alg.	*Conv. Ep.*
S(λ)	**8.52**
Q	13.8
Q(λ)	12.24
AC	11.6
DS(λ)	31.32
DQ	12.16
DQ(λ)	26.6
DAC	29.04

Table 2. Statistical significance, where $\star$ denotes $p < 0.0001$

Alg.	S(λ)	Q	Q(λ)	AC	DS	DQ	DQ(λ)	DAC
S(λ)	1							
Q	$\star$	1						
Q(λ)	$\star$	**0.0486**	1					
AC	**0.0028**	**0.0352**	0.4961	1				
DS(λ)	$\star$	$\star$	$\star$	$\star$	1			
DQ	**0.0032**	0.1804	0.9443	0.6724	$\star$	1		
DQ(λ)	$\star$	$\star$	$\star$	$\star$	0.1950	$\star$	1	
DAC	$\star$	$\star$	$\star$	$\star$	0.6120	$\star$	0.5355	1

As we can see in Table 1, SARSA(λ) outperforms all algorithms while enjoying statistically significant differences, as shown in Table 2, and DynaQ Learning outperforms the rest model based algorithms. SARSA(λ)'s performance can be in part explained by the eligibility traces that allow past experience to aid the learning process. Model based algorithms evidently do not perform that well, with the statistically insignificant differences (with $p > 0.05$) explained by the fact that these algorithms (including AC) are highly unstable and therefore have very high variance in terms of number of episodes required to converge. We implemented all algorithms in our system to allow the designer freedom of choice (each algorithm is representative of an RL class of methods) and flexibility in unforeseen issues.

5 Concluding Remarks and Future Work

As we can see from Dialogue 1 and 2, there is a clear improvement on the system's behaviour during learning and the optimal dialogue policy resembles one that a human designer would use. While this is a very simple scenario, one can imagine that in a system with many more slots and admissible queries, handcrafted policies are very hard or even impossible to create and inflexible when coming to adapting to users' needs and to abrupt changes in their goals. Online RL can deal with such problems and scale to real world applications thus alleviating the need of huge and complicated handcrafted policies. As mentioned before, our system provides the option of importing a handcrafted policy, that represents prior knowledge that the system may refine to its current needs.

In the future we plan to implement state of the art online RL algorithms, such as Natural Actor Belief Critic [4]. We also plan to apply hierarchical RL to achieve a more natural representation of the system's available actions and the users' goals and also apply techniques such as Complex Action Learning. To this aim we will need to formulate the slot filling problem as a Semi Markov Decision Process (SMDP). SMDPs allow for temporal abstraction in the model, meaning an action can take an arbitrary amount of time to complete and so we can model complex actions. Last we plan to test the Museum Guide system with real users to gain valuable feedback and intuition.

References

1. Bohus, D., Rudnicky, A.I.: The RavenClaw dialog management framework: Architecture and systems. Computer Speech & Language 23(3), 332–361 (2009)
2. Cuayáhuitl, H., Renals, S., Lemon, O., Shimodaira, H.: Evaluation of a hierarchical reinforcement learning spoken dialogue system. Comput. Speech Lang. 24, 395–429 (2010)
3. Gašić, M., Jurčíček, F., Thomson, B., Yu, K., Young, S.: On-line policy optimisation of spoken dialogue systems via live interaction with human subjects. In: Automatic Speech Recognition and Understanding, Hawaii (2011)
4. Jurčíček, F., Thomson, B., Keizer, S., Mairesse, F., Gašić, M., Yu, K., Young, S.: Natural Belief-Critic: A Reinforcement Algorithm for Parameter Estimation in Statistical Spoken Dialogue Systems. International Speech Communication Association 7, 1–26 (2010)
5. Konstantopoulos, S.: An Embodied Dialogue System with Personality and Emotions. In: Proceedings of the 2010 Workshop on Companionable Dialogue Systems, ACL 2010, pp. 31–36 (2010)
6. Peng, J., Williams, R.: Incremental multi-step Q-Learning. Machine Learning, 283–290 (1996)
7. Pietquin, O., Geist, M., Chandramohan, S., Frezza-Buet, H.: Sample-Effcient Batch Reinforcement Learning for Dialogue Management Optimization. ACM Transactions on Speech and Language Processing 7(3), No. 7 (2011)
8. Pietquin, O., Hastie, H.: A survey on metrics for the evaluation of user simulations. The Knowledge Engineering Review (2011) (to appear)
9. Rieser, V., Lemon, O.: Natural Language Generation as Planning Under Uncertainty for Spoken Dialogue Systems. In: EACL 2009, pp. 683–691 (2009)
10. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, Cambridge (1998)
11. Szepesvári, C.: Algorithms for Reinforcement Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 4(1), pp. 1–103. Morgan & Claypool Publishers (2010)
12. Watkins, C.J.C.H.: Learning from delayed rewards, PhD Thesis, University of Cambridge, England (1989)
13. Wiering, M.A., Van Hasselt, H.: The QV family compared to other reinforcement learning algorithms. In: IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, pp. 101–108 (2009)
14. Young, S., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B., Yu, K.: The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management. Computer Speech & Language 24(2), 150–174 (2010)