Machine Learning for Spoken Dialogue Management: an Experiment with Speech-Based Database Querying

Olivier Pietquin[1]

Supélec – Campus de Metz, rue Edouard Belin 2,
F-57070 Metz – France
olivier.pietquin@supelec.fr

Abstract. Although speech and language processing techniques achieved a relative maturity during the last decade, designing a spoken dialogue system is still a tailoring task because of the great variability of factors to take into account. Rapid design and reusability across tasks of previous work is made very difficult. For these reasons, machine learning methods applied to dialogue strategy optimization has become a leading subject of researches since the mid 90's. In this paper, we describe an experiment of reinforcement learning applied to the optimization of speech-based database querying. We will especially emphasize on the sensibility of the method relatively to the dialogue modeling parameters in the framework of the Markov decision processes, namely the state space and the reinforcement signal. The evolution of the design will be exposed as well as results obtained on a simple real application.

Keywords: Spoken Dialogue Systems, Reinforcement Learning, Dialogue Management.

1 Introduction

In the last few years, research in the field of Spoken Dialogue Systems (SDS) has experienced increasing growth. But, the design of an efficient SDS does not basically consist in combining speech and language processing systems such as Automatic Speech Recognition (ASR) and Text-to-Speech (TTS) synthesis systems. It requires the development of an interaction management strategy taking at least into account the performances of these subsystems (and others), the nature of the task (i.e. form filling or database querying) and the user's behavior (i.e. cooperativeness, expertise). The great variability of these factors makes rapid design of dialogue strategies and reusability across tasks of previous work very complex. For these reasons, machine learning techniques applied to strategy optimization is currently a leading domain of

[1] This work was realized when the author was with the Faculty of Engineering, Mons (FPMs, Belgium) and was sponsored by the 'Direction Générale des Technologies, de la Recherche et de l'Énergie' of the Walloon Region (Belgium, First Europe Convention n° 991/4351).

researches. When facing a learning problem, two main classes of methods could be envisioned: supervised and unsupervised learning. Yet, supervised learning would require examples of ideal (sub)strategies which are typically unknown. Indeed, no one can actually provide an example of what would have objectively been the perfect sequencing of exchanges after having participated to a dialogue. Humans have a greater propensity to criticize what is wrong than to provide positive proposals. In this context, Reinforcement Learning (RL) [1] appears as the best solution to the problem and have been first proposed in [2] and further developed in [3][4][5]. The main differences between the approaches rely in the way they model the dialogue manager's environment during the learning process. In [2] and [5], the environment is modeled as a set of independent modules (i.e. ASR system, user) processing information (this approach will be adopted in this paper). In [3], the environment is modeled as a pure state-transition system for which parameters are learned from dialogue corpora. In [4], a hybrid approach is described.

However, transposing spoken dialogue management in the formalism of the Markov Decision Processes (MDP) is required in every approach and it is probably the crucial point. No rational method but common sense is generally used to define the parameters of the corresponding MDP. In this paper, we emphasize on the importance of these parameters. The sensible steps are mainly the state space definition and the choice of the reinforcement signal. This is demonstrated on a simple speech-based database querying application.

2 Markov Decision Processes and Dialogue Management

2.1 MDP and Reinforcement Learning

In the MDP formalism, a system is described by a finite or infinite number of states $\{s_i\}$ in which a given number of actions $\{a_j\}$ can be performed. To each state-action pair is associated a transition probability $\mathcal{T}$ giving the probability of stepping from state *s* at time *t* to state *s*' at time *t*+1 after having performed action *a* when in state *s*. To this transition is also associated a reinforcement signal (or reward) r_{t+1} describing how good was the result of action *a* when performed in state *s*. Formally, an MDP is thus completely defined by a 4-tuple $\{S, A, \mathcal{T}, \mathcal{R}\}$ where *S* is the state space, *A* is the action set, $\mathcal{T}$ is a transition probability distribution over the state space and $\mathcal{R}$ is the expected reward distribution. The couple $\{\mathcal{T}, \mathcal{R}\}$ defines the dynamics of the system:

$$\begin{aligned}\mathcal{T}_{ss'}^{a} &= P\left(s_{t+1} = s' \mid s_t = s, a_t = a\right)\\ \mathcal{R}_{ss'}^{a} &= E\left[r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\right]\end{aligned} \quad (1)$$

These last expressions assume that the Markov property is met, which means that the system's functioning is fully defined by its one-step dynamics and that the functioning from state *s* will be identical whatever the path followed until *s*. To control a system described as an MDP (choosing actions to perform in each state), one would need a *strategy* or *policy* π mapping states to actions: $\pi(s) = P(a|s)$ (or $\pi(s) = a$ if the strategy is deterministic).

In this framework, a RL *agent* is a system aiming at optimally mapping states to actions, that is finding the best strategy π^* so as to maximize an overall reward R which is a function (most often a weighted sum) of all the immediate rewards r_t. If the probabilities of equations (1) are known, an analytical solution can be computed by dynamic programming [1], otherwise the system has to learn the optimal strategy by a trial-and-error process. RL is therefore about how to optimally map situations to actions by trying and observing *environment*'s feedback. In the most challenging cases, actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. Trial-and-error search and delayed rewards are the two main features of RL. Different techniques are described in the literature, in the following (mainly in section 4) the Watkin's Q(λ) algorithm [1] will be used.

2.2 Dialogue Management as an MDP

As depicted on Fig.1, a task-oriented (or goal-directed) man-machine dialogue can be seen as a turn-taking process in which a human user and a Dialogue Manager (DM) exchange information through different channels processing speech inputs and outputs (ASR, TTS ...). In this application, the DM *strategy* has to be optimized and the DM will be the learning *agent.* Thus the *environment* modeled by the MDP comprises everything but the DM: the human user, the communication channels (ASR, TTS …), and any external information source (database, sensors etc.). In this context, at each turn *t* the DM has to choose an *action* a_t according to its interaction *strategy* so as to complete the task it has been designed for. These actions can be greetings, spoken utterances (constraining questions, confirmations, relaxation, data presentation etc.), database queries, dialogue closure etc. They result in a response from the DM environment (user speech input, database records etc.), considered as an observation o_t, which usually leads to a DM *internal state* update. To fit to the MDP formalism, a *reinforcement signal* r_{t+1} is required. In [3] it is proposed to use the contribution of an action to the user's satisfaction. Although this seems very subjective, some studies have shown that such a reward could be approximated by a linear combination of objective measures such as the duration of the dialogue, the ASR performances or the task completion [6]. A practical example will be provided subsequently.

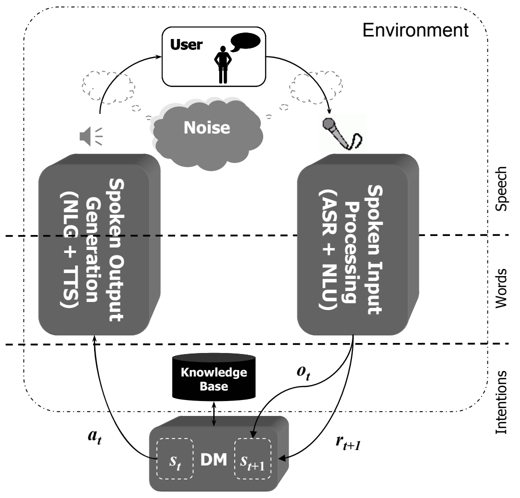

Fig. 1. Dialogue management as an MDP. NLG stands for Natural Language Generation and NLU stands for Natural Language Understanding.

3. Speech-Based Database Querying

The considered task consists in a database querying system. The goal of the application is to present a list of computers selected in a database according to specific features provided by a user through speech-based interaction.

The database contains 350 computer configurations split into 2 tables (for notebooks and desktops), each of them containing 6 fields: pc_mac (pc or mac), processor_type, processor_speed, ram_size, hdd_size and brand.

3.1 Action Set

Since the task involves database querying, actions will not only imply interaction with the user (such as spoken questions, confirmation requests or assertions) but also with the database (such as database querying). The action set contains 8 generic actions:

- greet: greeting (e.g. "How may I help you ?").
- constQ(*arg*): ask to constrain the value of *arg*.
- openQ: ask an open ended question.
- expC(*arg*): ask to confirm the value of *arg*.
- allC: ask for a confirmation of all the arguments.
- rel(*arg*): ask to relax the value of *arg*.
- dbQ([*args*]): perform a database query thanks to retrieved information.
- close: present data and close the dialogue session.

The value of *arg* may be the table's type (notebook or desktop) or one of the 6 table fields. Notice that there is not data presentation action because it will be considered that the data presentation is included in the 'close' action.

3.2 State Space

Building the state space is a very important step and several state spaces can be envisioned for the same task. Yet, some general considerations might be taken into account:

1. The state representation should contain enough information about the history of the dialogue so the Markov property can be assumed.
2. State spaces are often considered as informational in that sense that they are built thanks to the amount of information the DM could retrieve from the environment until it reached the current state.
3. The state representation must embed enough information so as to give an accurate representation of the situation to which an action has to be associated (it is not as obvious as it sounds).
4. The state space must be kept as small as possible since the RL algorithms converge in linear time with the number of states of the underlying MDP.

According to these considerations and the particular task of database querying, two slightly different state spaces where built to describe the task as an MDP to illustrate the sensitivity of the method to the state space representation. In the first representation, referred to as S_1 in the following, each state is represented by two features.

- A vector of 7 boolean values [f_x] (one for each value of *arg*). Each of these f_x is set to *true* if the corresponding value of *arg* is known (for example if the user specified to search in the notebooks table, f_0 is set to *true*). This is a way to meet the Markov property (informational state).
- Information about the Confidence Level (CL) of each f_x set to *true*. The confidence level is usually a real number ranging between 0 and 1 computed by the speech and/or language analysis subsystems (ASR and NLU) and providing information about the confidence of the system in the result of its processing. To keep the size of the state space reasonable, we only considered 2 possible values for the confidence level: *High* or *Low* (i.e. *High* means CL $\geq$ 0.8 and *Low* means CL < 0.8).

Notice that 'dbQ' actions will only include values with a *High* confidence level. For each value of *arg*, there are 3 different possibilities for the corresponding slot in the state representation: $\{f_x = false, CL = undef\}$, $\{f_x = true, CL = Low\}$, $\{f_x = true, CL = High\}$. This leads to 3^7 possible states.

The second state representation is built on the same basis but an additional state variable *NDB* is added to take the number of records returned by the last 'dbQ' action into account. This variable can also take only two values (*High* or *Low*) and is set according to the comparison of the query result size and a predefined threshold. If no 'dbQ' action has been performed, the *NDB* variable is initialized with the *High* value (an empty query would provide the whole database as a result). This state space representation will be referred to as S_2 in the following.

3.3 Reward Function

Again, several proposals can be made for building the reward function and slight differences in the choices can result in large variations in the learned strategy. To illustrate this, some simple functions will be described in the following. According to [6], the reward function (which is here a cost function that we will try to minimize) should rely on an estimate of the dialogue time duration (*D*), the ASR performances (*ASR*) and the task completion (*TC*) so as to approximate the user's satisfaction using objective measures:

$$R = w_D \cdot D - w_{ASR} \cdot ASR - w_{TC} \cdot TC \tag{2}$$

In this last expression, the w_x factors are positive weights. Considering the estimate of the time duration, two values are actually available: the number of user turns $D = N_U$ (the number of turns perceived by the user) and the number of system turns $D = N_S$ (including database queries as well).

On another hand, the task completion is not always easy to define. The *kappa* coefficient defined in [6] is one possibility but didn't always prove to correlate well with the perceived task completion. For the purpose of this experiment, two simple task completion measures will be defined:

$$TC_{\max} = \max(\#(G_U \cap R)) \tag{3}$$

$$TC_{av} = average(\#(G_U \cap R)) \tag{4}$$

In these last expressions $\#(G_U \cap R)$ is the number of common values in the user's goal G_U (the user goal is supposed to have the same structure as an existing database record and is set before the dialogue begins) and one of the records *R* presented to the user at the end of a dialogue. When a value is not present in the user goal it is considered as common (if a field is not important to the user, it is supposed to match any value). The first task completion measure TC_{max} indicates how close the closest record in the presented results is. The second TC_{av} measures the mean number of common values between the user's goal and each presented record.

Finally, the ASR performance measures will be provided by the confidence levels (*CL*) computed by the ASR system after each speech recognition task.

4. Experiments

The number of required interactions between a RL agent and its environment is quite large (10^4 dialogues at least in our case). So, it has been mandatory to simulate most of the dialogues for two main reasons. First, it is very difficult (and expansive) to obtain an annotated dialogue corpus of that size. Second, it would have been too time consuming to realize this amount of spoken dialogues for training. So, in a first approximation, a written-text-based simulation environment has been built [5]. It simulates ASR errors using a constant Word Error Rate (WER) and provides confidence levels according to a distribution measured on a real system. If the system has to recognize more than one argument at a time, the *CL* is the product of individual *CL*s obtained for each recognition task (so it decreases). Other ASR simulation models can be considered [7] but it is out of the scope of this paper.

Several experimental results obtained with different settings of the state space and the reward function will be exposed in the following. These settings are obtained by combining in three different ways the parameters S_1, S_2, N_U, N_S, TC_{max}, TC_{av} mentioned before. Results are described in terms of average number of turns (user and system), average task completion measures (TC_{max} and TC_{av}) for the performance and in terms of action occurrence frequency during a dialogue session to get a clue about the learned strategy. These results are obtained by simulating 10,000 dialogues with the learned strategy.

4.1 First Experiment: S_1, N_U, TC_{max}

The first experiment is based on the smaller state space S_1 (without any clue about the number of retrieved records). The dialogue cost is computed thanks to the number of user turns N_U as a measure of the time duration and the TC_{max} value as the task completion measure. Results are as follows:

Table 1: Performances of the learned strategy for the $\{S_1, N_U, TC_{max}\}$ configuration

N_U	N_S	TC_{max}	TC_{av}
2.25	3.35	6.7	1.2

Table 2: Learned strategy for the $\{S_1, N_U, TC_{max}\}$ configuration

greet	constQ	openQ	expC	AllC	rel	dbQ	close
1.00	0.06	0.0	0.14	0.0	0.05	1.10	1.00

When looking at the three first columns of the performance table, the learned strategy doesn’t look so bad. It actually has a short duration in terms of user turns as well as in system turns and has a very high task completion rate in terms of TC_{max} measure. Yet the TC_{av} shows a very low mean value.

When looking to the average frequency of actions in table, one can see that the only action addressed to the user that happens frequently during a dialogue is the greeting action. Others almost never happen. Actually, the learned strategy consists in uttering the greeting prompt to which the user should answer by providing some argument values. Then the system performs a database query with the retrieved attributes and provides the results to the user. Sometimes, the user doesn't provide any attribute when answering to the greeting prompt or the value is not recognized at all by the ASR model, so the strategy is to perform a constraining question (and not an open ended question) that will provide an argument with a better *CL*. Sometimes the provided arguments have a poor *CL* and an explicit confirmation is asked for . Sometimes the provided arguments don't correspond to any valid record in the database so the strategy is to ask for relaxation of one argument (this also explains why the number of database queries is greater than 1). The value of TC_{max} is not maximal because sometimes the dialogue fails.

This results in presenting almost all the database records when the user only provides one argument when prompted by the greeting. This is why there is a so big difference between TC_{max} and TC_{av}. The desired record is actually in the presented data (TC_{max} is high) but is very difficult to find (TC_{av} is low). The learned strategy is definitely not suitable for a real system.

4.2 Second Experiment: S_2, N_U, TC_{av}

This experiment uses the same settings as the previous one except that the *NDB* variable is added to the state variables and the task completion is measured with TC_{av}. Results are as follows:

Table 3: Performances of the learned strategy for the {S_2, N_U, TC_{av}} configuration

N_U	N_S	TC_{max}	TC_{av}
5.75	8.88	6.7	6.2

Table 4: Learned strategy for the {S_2, N_U, TC_{av}} configuration

greet	constQ	openQ	expC	AllC	rel	dbQ	close
1.00	0.87	1.24	0.31	1.12	0.21	3.13	1.00

This time, TC_{max} and TC_{av} are close to each other, showing that the presented results are more accurate but the number of turns has increased. The number of system turns particularly shows higher values. This observation is obviously explained by the increase of database queries.

Looking at the action occurrence frequencies one can see that the learning agent tries to maximize the TC_{av} value while minimizing the number of user turns and maximizing recognition performance. To do so, it always performs a database query after having retrieved information from the user. Since the number of results is part of the state representation, the agent learned not to present the results when in a state with a high number of results. If this number is too high after the greeting, the learner tries to reach a state where it is lower. Thus it almost systematically performs an 'openQ' action after the greeting in order to get as much information as possible in a minimum of turns (this explains the 1.24 value). Yet, this often results in poorer recognition outputs, thus it also performs a confirmation of all the fields before presenting any result. Sometimes, more information is provided after the greeting and only a constraining question is needed to gather enough information to reach a state with less result. A constraining question is preferred in this case because it leads to better recognition results.

The mean number of user turns shows that only 5.75 turns are usually needed to reach an accurate result set because the computer configurations are sufficiently different so as not to need too much attributes in the database query to provided accurate results. Thus, the system doesn't ask for all the attribute values to the user. Further investigations would show that the system takes advantage of the structure of the database and asks for attributes allowing extracting the desired records as fast as possible.

4.3 Third Experiment: S_2, N_S, TC_{av}

The same experiment as the previous one has been performed but replacing the N_U measure of time duration by the N_S measure. It actually makes sense since in a real application, the database could be much larger than the one used here. Thus, the database queries could be much more time consuming.

Table 5: Performances of the learned strategy for the {S_2, N_S, TC_{av}} configuration

N_U	N_S	TC_{max}	TC_{av}
6.77	7.99	6.6	6.1

Table 6: Learned strategy for the {S_2, N_S, TC_{av}} configuration

greet	constQ	openQ	expC	AllC	rel	dbQ	close
1.00	1.57	1.24	0.33	1.32	0.31	1.22	1.00

This obviously results in a decrease of the number of database queries involving a proportional decrease of the number of system turns N_S. Yet, an increase of the number of user turns N_U is also observed. By examining the action frequencies, one can notice that the number of constraining questions increased resulting in an increase of N_U. Indeed, the learned strategy implies gathering enough information from the user before performing a database query. This explains why the systems ask more constraining questions.

This last strategy is actually optimal for the considered simulation environment (constant word error rate for all tasks) and is suitable for using with this simple application.

5. Conclusion

In this paper, we first described the paradigms of the Markov Decision Processes (MDP) and of Reinforcement Learning (RL) and explained how they could be used for spoken dialogue strategy optimization. Although RL seems suitable for this task, the parameterization of such a method influences a lot the results and is very task dependent. We therefore wanted to show by three experiments on a very simple database querying system the influence of parameterization. From this, one can say first that the state space representation is a crucial point since it embeds the knowledge of the system about the interaction. Second, the reward (or cost) function is also of major importance since it measures how well the system performs on the task. Performance measure is a key of RL. The three experiments described in the last section showed the influence of these parameters on the learned strategy and concluded that a correctly parameterized RL algorithm could result in an acceptable dialogue strategy while little changes in the parameters could lead to silly strategies unsuitable for use in real conditions.

References

1. Sutton, R. S., Barto, A.G.: Reinforcement Learning : An Introduction. MIT Press (1998)
2. Levin, E., Pieraccini, R., Eckert, W.: Learning Dialogue Strategies within the Markov Decision Process Framework. Proceedings of ASRU'97, Santa Barbara, California (1997)
3. Singh, S., Kearns, M., Litman, D., Walker, M.: Reinforcement Learning for Spoken Dialogue Systems. Proceedings of NIPS'99, Denver, USA (1999)
4. Scheffler, K., Young, S.: Corpus-Based Dialogue Simulation for Automatic Strategy Learning and Evaluation. Proceedings of NAACL Workshop on Adaptation in Dialogue Systems (2001)
5. Pietquin, O., Dutoit, T.: A Probabilistic Framework for Dialog Simulation and Optimal Strategy Learning. IEEE Transactions on Audio, Speech and Language Processing, Volume 14, Issue 2 (2006) 589-599.
6. Walker, M., Litman, D., Kamm, C., Abella, A.: PARADISE: A Framework for Evaluating Spoken Dialogue Agents. Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, Madrid, Spain (1997) 271-280.
7. Pietquin, O., Beaufort, R.: Comparing ASR Modeling Methods for Spoken Dialogue Simulation and Optimal Strategy Learning. Proceedings of Interspeech/Eurospeech 2005, Lisbon, Portugal (2005)