FAST REINFORCEMENT LEARNING OF DIALOG STRATEGIES

*David Goddeau and Joelle Pineau**

Compaq Computer Corporation
Cambridge Research Laboratory
One Kendall Square, Building 700
Cambridge, Massachusetts 02139
United States

ABSTRACT

Dialog management is a critical component of an effective spoken language application. It is also one of the most difficult and time consuming to engineer. This paper examines the application of reinforcement learning and Markov Decision Processes (MDP's) to the problem of learning the dialog strategies. It extends work done at AT&T [1] [2] in two directions. First it examines the ability of RL to learn optimal strategies in the presence of speech recognition errors. Second, it describes a technique for reducing the amount of data required to train these models. This is significant as the difficulty of training MDP-based dialog managers is a serious roadblock to deploying them in realistic applications.

1. INTRODUCTION

Dialog management is a critical component of an effective spoken language application. The task of the dialog manager is, in essence, to decide what to say or do next, given the current context and user input. This decision can be complex, even for the simplest applications such as form-filling, due to recognition errors, unreliable channels, and naive, impatient, or un-cooperative users. For applications which incorporate natural language understanding, conversational style, mixed initiative, or more complex goals, the problem of dialog strategy design becomes much more difficult.

One can formalize the notion of a dialog as a sequence of alternating utterances $M_0, U_0, M_1, U_1, ...M_i, U_i, ...M_N$ in which the M_i and U_i are elements of a set of utterances, and specify the machine's and user's turns respectively. Given this notion of a dialog, a dialog manager is a policy function, π which selects M_i based on the previous dialog context and the internal knowledge base (KB) of the system.

$$M_i = \pi(M_0, U_0, ...M_{i-1}U_{i-1}; KB) \quad (1)$$

This paper examines the application of reinforcement learning and Markov Decision Processes (MDP's) to the problem of learning the dialog policy function π defined above. In particular, we extend work done at AT&T [1] [2] to deal with recognition errors and we present methods to reduce the amount of training data required, a limiting factor in applying MDP's to dialog management. Section 2 briefly describes Markov Decision Processes and their application to dialog management. Section 3 describes experiments evaluating the ability of the MDP to learn dialog strategies in the context of unknown recognition error rates. Section 4 addresses the problem of reducing the amount of data needed to train MDP dialog models.

2. MDP'S AND REINFORCEMENT LEARNING

A Markov Decision Process is characterized by four elements

- a set of states $s \in S$,
- a set of actions $a \in A$,
- a set transition probabilities $T(s, a, s')$ (= Probability of transitioning to state s' from state s having taken action a),
- and a reward distribution $R(s, a, r)$ (= probability of receiving reward r from state s having taken action a).

Given a completely specified MDP, several algorithms exist (see [3]) which compute for each state, the expected total (discounted) future reward for taking each action. This immediately determines the optimal action to take from each state, (the one with the greatest expected reward) and therefore the optimal policy for the system.

In most applications, however, the model parameters are not known and must be learned through experience. There are many approaches to this problem which vary primarily in what function is learned (ie the expected reward for each state-action or the underlying model probabilities) and how the state space is to the explored (see [3] for an overview).

In the work described here, the system learns the underlying model probabilities $T(s, a, s')$ and $R(s, a)$. The expected rewards and optimal policy are then computed based on these parameters using dynamic programming. This approach is practical for dialog applications as the amount of training data, not computation, is the limiting resource.

* Joelle Pineau is a PhD student at Carnegie Mellon University. This work was performed during a summer intern-ship at Cambridge Research Laboratories, MA.

2.1. Reinforcement Learning of Dialog Strategies

Markov Decision Processes were first applied to the problem of dialog management by Pieraccini and Levin at AT&T [1]. They later used reinforcement learning techniques to learn dialog strategies for model dialog problems [2]. While successfully learning reasonable dialog strategies, this work highlighted one of the problems of this method, the large amount of training data required. In [2], as in the work reported here, a synthetic user model was constructed to interact with the system during the training process. Additional work at AT&T [4] applied data from human-human dialogs to MDP dialog models. This paper extends this work in two directions: first it explores the ability of reinforcement learning techniques to learn effective strategies in the context of unknown factors, such as recognition error rate, and second, it presents methods for reducing the amount of training data required for learning.

3. LEARNING DIALOG STRATEGIES FOR IMPERFECT RECOGNITION

Dialog strategy design for form-filling applications, which generally do not require much natural language understanding, would be straightforward given perfect speech recognition and an experienced, patient, and co-operative user. The success of form-based Web applications for stock trading, travel planning, etc. is evidence for this. However, in spoken dialog applications, speech recognition has a significant error rate and this is one of the factors which complicate dialog design for real applications. Although it is straightforward to prompt for a series of data items and record the recognized responses, in practice some fraction of this recognized data will be incorrect. The correctness of an item, or list of items can be verified by asking the user, but this extends the length of the dialog and can annoy the user with constant verification requests.

3.1. MDP Model of Form-Filling Dialog

In order to map dialog tasks onto the MDP model, an appropriate set of states and actions must be selected. The states must be chosen to capture the essential aspects of the dialog context without requiring an unworkably large state space. For these experiments, the state space was chosen to represent the system's progress in the form-filling task. This ignores some important aspects of the dialog, for example, how many times a given prompt has been repeated, but is sufficient for the purposes of these experiments. Each state is a composition of several fields, a status field per data slot with values, `Empty`, `Filled`, or `Verified`, and an additional field indicating whether the system is in a final state. Note that the MDP can only track whether a given slot has been filled or verified. It cannot tell whether the particular values are correct or not.

The set of actions (prompt types) the system can use are:

- 'Request All' - Request the user fill all slots in one response.
- 'Request n' - Request the user give the data for slot n,

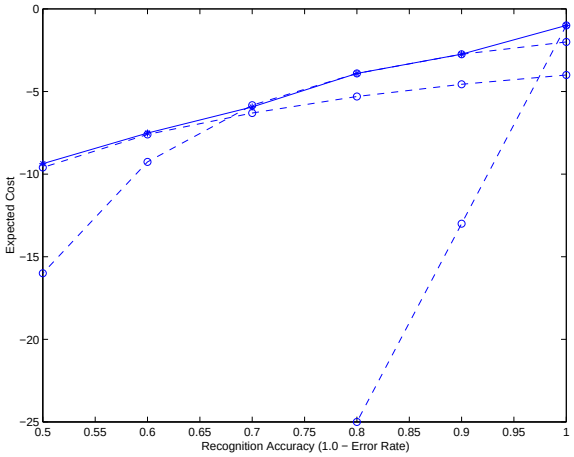

Figure 1: Expected Cost for Several Dialog Strategies as Function of Recognition Accuracy. Solid Line Represents Behavior of Reinforcement Learning System

- 'Verify n' - Ask the user if the (previously acquired) value for slot n is correct
- 'Verify All' - Ask the user to verify all slot values. (a negative response implies at least one item is wrong, but does not specify which).
- 'Quit' - Conclude the dialog.

3.1.1. Reward Structure

Costs (rewards) are presented to the system after each action. The cost function charges the system a fixed amount for every prompt to the user, and a final cost for each unfilled or incorrectly filled slot. The goal of the system is to minimize the expected cost per dialog. For example, given a perfect recognizer and co-operative user, the optimal strategy would be to Request All, then Quit, as all slots would be correctly filled using the minimum number of prompts. However, this strategy will not be optimal for an error prone recognizer as one or more of the returned values may be incorrect. The relative costs of asking a question and returning an incorrect value were chosen to strongly discourage returning incorrect values.

3.2. User Model

Since this experiment was designed to examine the ability of reinforcement learning to produce optimal dialog strategies, a synthetic user model was used to generate training episodes. In this way, the recognition error rate and user behavior can be varied in controlled fashion. For this experiment, the user was set to be co-operative (ie completely answering each question), and the recognition accuracy was varied from 50% to 100%. For each error rate, the system was trained to convergence and the resulting expected cost and dialog policy were noted. The results are presented in Figure 1.

3.3. Results and Discussion

The solid line in Figure 1 plots the expected cost vs. recognition accuracy for the learned behavior of the MDP. As the error rate increases the system adopts one of 3 basic strategies, that can be roughly described as follows.

- 1) No verify. – The system trusts the data is correct.
- 2) Verify all. – The system tries to verify all slots at once. If there is an error, it starts over from the beginning.
- 3) Verify individually. – The system verifies each slot individually, if the data is in error, it requests new data.

The cost vs. accuracy tradeoffs of these strategies are plotted with broken lines in Figure 1. While each of these strategies is optimal for some range of recognition accuracy, none is optimal over the entire range. In all strategies, the system initially requests all slot data, then requests individually any data items that were tagged incorrect by the verification process. As shown in the graph, for each error rate, the trained system chooses the optimal strategy. It should be noted that the system does not explicitly know the recognition error rate, but merely learns the expected reward associated with each state.

The model described above could be extended in several ways to be more realistic. For example, the response of the user to "Request All" prompts could be varied to only return a subset of slot values, or the user could incorporate corrections into their response to Verify requests. or the recognition error rate could be varied on a slot by slot basis. However, none of these affect the fundamental structure or behavior of the system, they merely change the state transition probabilities. In addition, confidence measures could be added to the recognition response. Again, this does not change the system fundamentally, but multiplies the number of states as an additional Confidence field is added to the state description. This in turn increases the number of training episodes required.

4. FASTER TRAINING OF MDP DIALOG MODELS

One of the drawbacks of using MDP's to learn dialog strategies is the large number of training dialogs required. The number of training dialogs needed is in general exponential in the number of states, which is itself exponential in the complexity of the dialog task (number of form slots to be filled). This section describes an approach which can drastically reduce the number of training episodes required for certain types of applications.

During exploration, the MDP is attempting to learn the expected reward for each state, on which it bases its policy. Alternatively, the system can learn the transition and reward distributions $T(s, a, s')$ and $R(s, a)$ the expected rewards computed via dynamic programming. In many RL applications, learning can be accelerated by using functional approximation techniques to estimate the expected reward function. These techniques, however, are restricted to state spaces on which a convenient distance metric between states can be found (for example, where the state space represents actual 2D space as in robot navigation applications). This is generally not the case for dialog state spaces.

We describe here an approach that can take advantage of the structure of dialog state spaces to estimate the transition probabilities $T(s, a, s')$ more efficiently. The basic idea is to adapt the techniques of back-off or interpolated models used in statistical language model estimation to the estimation of MDP transition probabilities.

The direct estimate of the transition probabilities are computed as

$$T_d(s, a, s') = \frac{Count(s, a, s')}{Count(s, a)}. \quad (2)$$

In general, a large number of training episodes are required before enough counts are accumulated to make this estimate reliable. To compensate for this, we associate with each state s a back-off state $bo(s)$ and with each action, a back-off action $bo(a)$. This mapping is many to 1, so several state share a common back-off state. As training proceeds, the system not only maintains the estimates T_d but also the back-off estimates

$$T_{bo}(s, a, s') = \frac{Count(bo(s), bo(a), bo(s'))}{Count(bo(s), bo(a))}. \quad (3)$$

Since the mappings from state and actions to their respective back-offs are many to 1, these back-off estimates are trained much more rapidly than the estimates for the individual states.

During operation, the transition distribution used for a given state,action pair is a combination of the direct estimate $T_d(s, a, s')$ and the back-off estimate $T_{bo}(s, a, s')$. There are several methods of combining these estimates including interpolation.

$$T(s, a, s') = \lambda * T_{bo}(s, a, s') + (1 - \lambda) * T_d(s, a, s') \quad (4)$$

or backing off,

$$T(s, a, s') = T_{bo}(s, a, s') \quad (5)$$

if $Count(s, a) < Thresh$ else

$$T(s, a, s') = T_d(s, a, s'). \quad (6)$$

In the experiment described below we have explored only the latter.

4.1. Using Back-Off Models in Form-Filling Dialog

The use of back-off states and actions allows an MDP model to take advantage of the structure of the state space to learn its parameters more quickly. The state space for form-filling tasks has a natural structure which can be exploited with back-off models. It particular, the sub-tasks of filling each slot are largely independent. The state space for this task can therefore be factored into a product of independent state spaces, which can be independently learned. In addition, the set of actions associated with each slot (Request n, Verify n) can be abstracted. This abstraction can be incorporated into the back-off state. This allows the MDP to apply the transition structure it learns about filling any slot, to filling every slot.

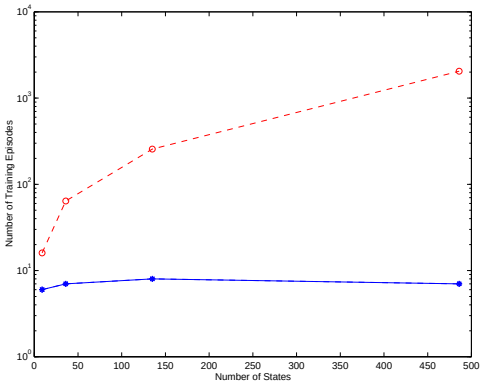

Figure 2: Plot of Training Data Required vs. State Space Size for Back-Off (solid line) and Non-back-off (broken line) Learning Models

To test the effectiveness of this technique, form-filling applications were trained with and without the back-off models to determine the relationship between the size of the model state space and the amount of training data required to reach peak performance. This experiment again uses an task-based state description for the form-filling application. For slot in the form there is an associated status value in the state description which can be `Empty`, `Filled`, or `Verified`. The basic state space for a n slot form is therefore at least 3^n states. The actions were those described above except the 'Verify All' and 'Request All' actions were disabled (to make the task harder). The cost structure was set to require that each slot be `Verified` as well as `Filled`. The user model for this experiment is similar to that described above, with a fixed recognition error rate.

The back-off model used in this experiment tries to exploit both the independence of each slot, and the similarity of structure among the slot-filling subtasks. The back-off state space was chosen to model a 1-slot form-filling task. Each state in the n-slot form-filling task that the system was trying to learn was assigned as its back-off state the closest corresponding state in the 1-slot task.

To measure the effect of using this back-off model on learning speed, several training runs were performed. The number of slots in the form, and therefore the size of the state space, was varied and the system was trained to convergence for each setting. Many training runs were performed for each setting and the average number of training utterances required was recorded. The results are plotted in Figure 2.

The upper curve in Figure 2 represents the amount of training data needed by the baseline system. This is exponential in the size of the state space which is itself exponential in the number of slots. The lower curve represents the amount of data required by back-off model, which is dramatically less. As the system explored the state space of the n-slot task, the state transition statistics were also accumulated for the corresponding back-off states. As a result, the system learned the transition probabilities for the back-off model very quickly. Since in this application, the back-off model is an excellent reflection of the base model, the system also learns the optimal policy very quickly.

5. CONCLUSIONS

This paper examined the application of reinforcement learning and Markov Decision Processes to learning dialog strategies. These techniques were shown be capable of learning optimal dialog strategies in the context of imperfect recognition. In addition, a back-off estimation method for learning model parameters was shown to dramatically reduce the amount of required training data.

Markov Decision Processes are a useful formal model of dialog management. However, several limitations must be overcome before MDP-based dialog managers can be effectively field in practice. One of the most critical is the amount of training data required. The back-off technique presented here, although very effective in some cases, is only a first step. In order to create natural and robust dialog managers, the model state space must capture many more aspect of the dialog structure. Dealing with this greatly expanded state space remains a problem. In order to address this, traditional reinforcement learning techniques must be augmented with other ideas, such as partial-order planning. In addition, exploration methods must be improved and integrated with normal system operation so the dialog manager can continue to learn and adapt to changing conditions. Finally, MDP implementations of dialog managers have so far ignored many problems of natural language processing focusing instead on high-level dialog strategy issues.

6. ACKNOWLEDGMENTS

We would like to thank Pedro Moreno for MatLab help and the Speech Group at CRL for support.

7. REFERENCES

[1] E. Levin and R. Pieraccini, "A stochastic model of computer-human interaction for learning dialog strategies," in *Proc. EUROSPEECH97, Rhodes, Greece*, 1997.

[2] E. Levin, R. Pieraccini, and W. Eckert, "Using markov decision process for learning dialog strategies," in *Proc. ICASSP*, 1998.

[3] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[4] M. Walker, J. Fromer, and S. Narayanan, "Learning optimal dialog strategies: A case study of a spoken dialog agent for email," in *Proc. ACL*, 1998.