INTERSPEECH 2009 BRIGHTON

# Reinforcement Learning for Dialog Management using Least-Squares Policy Iteration and Fast Feature Selection

*Lihong Li*[1], *Jason D. Williams*[2], *and Suhrid Balakrishnan*[2]

[1]Department of Computer Science, Rutgers University, Piscataway, NJ, 08854 USA
[2]AT&T Labs - Research, Building 103, 180 Park Avenue, Florham Park, NJ, 07932 USA
lihong@cs.rutgers.edu, jdw@research.att.com, suhrid@research.att.com

## Abstract

Reinforcement learning (RL) is a promising technique for creating a dialog manager. RL accepts *features* of the current dialog state and seeks to find the best action given those features. Although it is often easy to posit a large set of *potentially* useful features, in practice, it is difficult to find the subset which is large enough to contain useful information yet compact enough to reliably learn a good policy. In this paper, we propose a method for RL optimization which automatically performs feature selection. The algorithm is based on least-squares policy iteration, a state-of-the-art RL algorithm which is highly sample-efficient and can learn from a static corpus or on-line. Experiments in dialog simulation show it is more stable than a baseline RL algorithm taken from a working dialog system.

**Index Terms**: Dialog management, spoken dialog systems, partially observable Markov decision processes

## 1. Introduction

In a spoken dialog system, the dialog manager is the central component which decides what to say or do given the current dialog state. Unfortunately, because dialog is a temporal process, there are in general an astronomical number of dialog states. In industry, this problem is generally solved by manually designing an appropriately small space of compressed dialog states, for which a dialog manager can then be carefully designed by hand. While this has been used to build numerous deployed dialog systems, it can ignore potentially useful distinctions between dialog states and lead to sub-optimal dialog systems.

As a result, researchers have begun applying reinforcement learning (RL) techniques to automatically assign actions to dialog states [1]. The hope is that more dialog states can be considered, and thus more detailed dialog managers can be created, which for example better handle speech recognition errors. In practice the number of independent states that RL can consider is still bounded by practical limits of computational complexity and the size of available dialog corpora. As a result, researchers have suggested the dialog manager act on a *feature*-based representation of the dialog state – for example, [2, 3, 4, 5], among others. The hope is that features will enable the dialog manager to generalize even when the number of dialog states is massive.

Learning tractability now depends not on the number of possible dialog states, but on choosing a compact set of useful features about these states. Whereas it is often easy for a designer to suggest a large set of *potentially* useful features, it is difficult to decide which subset are *actually* useful for an RL algorithm. Thus there is an important feature selection problem facing the following bias-variance trade-off: on the one hand, using too few features ignores useful information which can improve dialog managers; on the other hand, using too many features prevents learning from converging within the limits of available data and computation time.

In this paper, we present a method for automatically selecting features for RL-based dialog management. The next section formalizes the problem and reviews related work; Section 3 introduces our method; Section 4 presents results; and Section 5 concludes.

## 2. Background and related work

To begin, we formalize the learning problem. The spoken dialog system's internal dialog state is $b$. This state includes the most recent automatic speech recognition (ASR) result, and also includes all the information the dialog system has collected about dialog history, such as values for fields, grounding status, the ASR confidence associated with a piece of information, distributions over latent user goals, and so on.

At each timestep in the dialog, the RL algorithm is given $b$, chooses an action $a$, and receives a reward $r$. Example actions include asking the user a question such as who they would like to call or confirming the name of a callee, consulting a database, or remaining silent. The reward $r$ is a real-valued measure of goodness of the current action and dialog state. A reward function $R$, which is a mapping $R(b, a) = r$, is specified by the human designer and allows the designer to specify trade-offs between quantities such as dialog speed and accuracy. The goal of the RL algorithm is to find a policy $\pi$ which is a mapping $\pi(b) = a$ that maximizes the cumulative rewards over the course of the entire dialog.

To choose actions it is useful to construct a value function $Q(b, a)$, which yields the expected sum of rewards for taking action $a$ in dialog state $b$, and subsequently continuing to follow the dialog manager's policy until the end of the dialog. However, the large cardinality of $b$ can make computing $Q(b, a)$ in tabular form hopeless. Instead, we extract $k$ features, $\boldsymbol{\phi}(b, a) = [\phi_1(b, a), \ldots, \phi_k(b, a)]^\top$, and use $\boldsymbol{\phi}(b, a)$ to estimate $Q$. These feature functions $\phi_i(b, a)$ are specified by the dialog system designer.

Feature-based RL for dialog management has been explored before. For example, Henderson et al. [3] use a linear approximation of $Q$, i.e., $Q(b, a) = \sum_i w_i \phi_i(b, a)$, where $w_i$ is a weight whose magnitude indicates the contribution of feature $i$ to the value function. Optimization is then performed with a modified SARSA algorithm (discussed more below). Williams [5] creates a distance metric $D(\boldsymbol{\phi}, \boldsymbol{\phi}')$ and clusters feature vectors to form template points, then performs standard value iteration on the template points.

All of these feature-based approaches face the problem of

*feature selection*: the designer must find a set of features which is on the one hand small enough to keep learning tractable, and on the other hand large enough to capture important distinctions to enable good policy learning. It is hard for designers to anticipate which features, and in which combinations, will be useful to RL algorithms. In the authors' experience, development of RL-based systems usually includes much trial-and-error searching for a compact but useful set of features, for example in developing a voice dialer [5].

The main contribution of this paper is to develop an RL algorithm which automatically selects among a large set of features. The motivation is that it is often relatively easy for a designer to propose a large set of *potentially useful* features; the difficulty is deciding on a compact subset which is *actually useful*. Although there are existing methods for feature selection in RL, they appear too expensive to be suitable for optimizing a realistic spoken dialog system [6, 7].

Our method is based on *least-squares policy iteration* (LSPI), a state-of-the-art general-purpose RL technique which has been successfully applied to a number of control and planning problems, such as riding a bicycle [8]. LSPI itself has not been applied to dialog management before, and is attractive starting point for a variety of reasons. First, it can learn either *on-policy* or *off-policy* – that is, it can learn either from a corpus generated with some other dialog manager, or learn when it is controlling the dialog. Other popular RL algorithms such as SARSA and Natural Actor Critic can only reliably learn online, when they are in control of the dialog manager. In past work which has used SARSA to learn from a corpus, modifications to the algorithm have been necessary which augment the reward function to penalize transitions to under-sampled regions of the feature space [3]. This in effect conflates the needs of learning with the dialog goals sought by the designer. LSPI requires no modification to the reward function, reserving its use for specification of dialog goals.

Second, LSPI is sample-efficient, making maximal use of data. Compared to other popular learning methods which support on-policy and off-policy learning such as Q-Learning, it learns better policies with less data [8]. This is important for dialog learning where example interactions are often in short supply. Moreover, LSPI never diverges, and completely avoids learning rate parameters required for algorithms like SARSA and Q-Learning. Removing hand-set parameters removes much of the "art" from getting an RL-based system working. Our extension adds two tunable parameters: one sets the fraction of the features to retain, and the other is a learning rate.

## 3. Method: LSPI-FFS

We will begin by adopting a standard linear approximation of the value function, $Q(b,a) = \sum_{i=1}^{k} w_i \phi_i(b,a)$, well-established in the RL literature and previously suggested for dialog management by Henderson et al [3]. The magnitude of a weight $w_i$ indicates the contribution of its feature $\phi_i(b,a)$ to the value. It is often helpful to standardize features (so that components in $\boldsymbol{\phi}$ have comparable value ranges), and we will require this here.

LSPI assumes a set of $m$ sampled transitions are provided: $\mathcal{D} = \{(b_1, a_1, r_1, b'_1), \ldots, (b_m, a_m, r_m, b'_m)\}$. Starting with an (arbitrary) initial weight vector $\mathbf{w}_1$ of dimension $k$, it improves this vector iteratively until it (almost) converges.

At iteration $j = 1, 2, \ldots$,

1. Let $Q_j(b,a) = \mathbf{w}_j^\top \boldsymbol{\phi}(b,a)$ be the current linear Q-function, and $\pi_j(b) = \text{argmax}_a Q_j(b,a)$ the corresponding greedy policy.
2. Obtain $Q_{j+1}(b,a)$ to approximate the Q-function of $\pi_j$: $Q_{j+1}(b,a) = \mathbf{w}_{j+1}^\top \boldsymbol{\phi}(b,a)$; here, $\mathbf{w}_{j+1}$ solves a system of linear equations, $\mathbf{A}\mathbf{w} = \mathbf{c}$, where $\mathbf{A}$ is a $k \times k$ matrix and $\mathbf{c}$ a $k$-vector computed using $\mathcal{D}$ by: $\mathbf{A} = \sum_{l=1}^{m} \boldsymbol{\phi}(b_l, a_l) \left(\boldsymbol{\phi}(b_l, a_l) - \gamma \boldsymbol{\phi}(b'_l, \pi_j(b'_l))\right)^\top$ and $\mathbf{c} = \sum_{l=1}^{m} \boldsymbol{\phi}(b_l, a_l) r_l$. The standard RL discount factor $\gamma \in (0,1)$ determines the present value of future rewards, and is specified by the designer [9].

If in step 2 above (known as the *LSTDQ* algorithm [8]) $Q_{j+1}$ computes *exactly* the Q-function of $\pi_j$, LSPI becomes *policy iteration* in which $\pi_{j+1}$ is always better than $\pi_j$, and the algorithm will converge to an optimal policy [9]. If $Q_{j+1}$ is a sufficiently good approximation, as we hope in our case, LSPI still converges to a near-optimal policy [8].

Note that LSPI in itself is difficult for large feature sets because of the solution to the dense linear system involved in LSTDQ (naively, with time complexity cubic in $k$). We thus develop a novel variant of LSPI to handle many features. First, observe that the magnitudes of weight $w_i$ in $\mathbf{w}$ indicate the relative strength of the contribution of $\phi_i$ to $Q$ (recall that we have standardized the features). Our basic idea is to retain only the features with strong contributions to the value function. The key insight is that an *approximation* of $\mathbf{w}$ which yields roughly the same *ordering* of magnitudes in $\mathbf{w}$ can be used to choose relevant features. Specifically, we propose to approximate $\mathbf{w}$ using a gradient-descent-like algorithm known as *temporal difference (TD)* [10] which has a time complexity linear in $k$ and converges to the same solution as LSTDQ in the limit. TD quickly computes a rough estimate of the weight vector $\hat{\mathbf{w}}$, from which a small subset of features is selected; then LSTDQ computes the expensive but exact weight vector $\mathbf{w}$ using this small feature subset. More precisely, step 2 of LSPI is replaced by:

2-1. Initialize $\hat{\mathbf{w}}$ to the zero vector, and run TD to go through the samples in $\mathcal{D}$ (possibly in multiple passes) to obtain $\hat{\mathbf{w}}$. For the sample $(b_l, a_l, r_l, b'_l)$, the TD update rule is $\hat{\mathbf{w}} \leftarrow \hat{\mathbf{w}} + \eta d_l \boldsymbol{\phi}(b_l, a_l)$, where $\eta \in (0,1)$ is a step size and $d_l = r_l + \gamma \hat{\mathbf{w}}^\top \boldsymbol{\phi}(b'_l, \pi_j(b'_l)) - \hat{\mathbf{w}}^\top \boldsymbol{\phi}(b_l, a_l)$ is the *temporal difference*.

2-2. Pick $k'$ ($\ll k$) features whose weights in $\hat{\mathbf{w}}$ are largest in magnitude.

2-3. Run LSTDQ (step 2 of LSPI) using the $k'$ features to obtain a new weight vector $\mathbf{w}_{j+1}$: the $k - k'$ components in $\mathbf{w}_{j+1}$ *not chosen* are set to 0, while the other $k'$ components are computed by LSTDQ.

We call the modified algorithm LSPI with Fast Feature Selection (LSPI-FFS). The parameter $k'$ in effect sets the resolution of the policy: increasing $k'$ adds resolution at the expense of requiring more training data. For the step size $\eta$, experimentation found that setting $\eta = 1/k$ appears to produce reliable results; this setting was used in the experiments below. One key property of the method is that the set of features selected may vary from one iteration to the next: as $Q$ is estimated for longer and longer planning horizons, different mixtures of features may be relevant.

## 4. Experiments

To test the method, we applied it to an existing voice dialer application, DIALER, which has been accessible within the AT&T research lab for several years and which receives daily calls. DIALER's vocabulary consists of about $50,000$ AT&T employees. Since many employees have the same name, DIALER can

disambiguate by asking for the callee's location. It can also disambiguate between multiple phone listings for the same person (office/mobile) and indicate when a callee has no number listed. This dialog system uses a hand-crafted dialog manager, which has processed thousands of calls successfully.

We use this dialog system as a harness to compare our method to two baseline methods for creating a dialog manager. One is the conventional hand-coded dialog manager currently running the dialog system. The second baseline is an RL algorithm which operates on features, but does not do any feature selection. It relies on a distance metric $D(\boldsymbol{\phi}, \boldsymbol{\phi}')$ to cluster together feature vectors into template points. It then estimates dynamics on these template points and performs standard value iteration. A complete description can be found in [5]. We feel this second baseline is strong because it is also designed for off-policy learning, has been demonstrated interactively to the research community [11], and has been active on a second dialer in the lab for about a year, processing real calls.

All of the RL methods reported below make use of a hand-crafted "co-controller" [12]. At each stage in the dialog, the co-controller nominates a *set* of plausible action types, and the RL algorithm chooses an action type from this set. Example action types include `AskName` which asks the callee's name, `AskPhoneType` which asks the intended phone type, `ConfirmName` which confirms the callee's name, and `TransferCall` which transfers the call. Past work has shown that using a co-controller is an effective way of incorporating domain knowledge and business rules into learning, and of increasing learning speed and reliability [12]. In the comparisons below, all of the RL algorithms are evaluated using the same hand-crafted co-controller.

For the reward function, the system receives a $-1$ penalty for every action it takes, a large final reward of $+20$ if the call is correctly transferred and a large penalty of $-20$ otherwise. This reward function reflects our desire for the system to ask the caller as few questions as possible and to transfer the call to the right person and phone type.[1] At most 20 actions are allowed per conversation. The discount factor is $0.99$.

In order to produce a large number of dialogs to test the methods, a user simulation was employed – the same user simulation which was used to train the RL Baseline previously demonstrated to the research community, giving us some confidence in its ability to fairly model user behavior. From labeled real dialogs with the existing (non-RL) dialer, probabilities over user actions were estimated, conditioned on dialog history, user goal, and the preceding system action. A synthetic corpus of $20,000$ dialogs was then generated by running the co-controller with the user simulation, choosing among available actions uniformly at random. The same synthetic dialogs were then used for optimization with each of the RL methods.

Past work developing the baseline RL algorithm explored a variety of features; through manual trial-and-error, the following set of features was found to perform best and are used here for the baseline RL algorithm. We call these *raw* features because they do not attempt to capture interactions between their components; in the experiments this feature set is referred to as the *hand-tuned raw* feature set. Parentheses indicate the type and cardinality of the features:

- Probability of most likely callee $p_c$ (continuous, 1)
- Probability of most likely phone type $p_t$ (continuous, 1)
- How many (0, 1, or 2) phone numbers are available for the most likely callee (discrete, 3)
- Whether the name of the most likely callee is ambiguous (discrete, 2)

In the development process, the first two features were obvious; the difficulty was determining how to express the information contained in the other two (discrete) features compactly. Our main claim is that the LSPI-FFS algorithm can effectively search among a large pool of easy-to-create features. We believe that features expressing *whether an action is available or not* form such a pool: when using the "co-controller approach", this information is readily available, and the set of available actions implies information about the system state. With $9$ action types, there are $512 = 2^9$ possible combinations of action availabilities, compared to the $6 = 2 \times 3$ discrete feature combinations above.

Thus, we created a second set of raw features by replacing the two discrete features above with one for each of the $9$ action types, yielding the *untuned raw* feature set with $11$ features:

- Probability of most likely callee $p_c$ (continuous, 1)
- Probability of most likely phone type $p_t$ (continuous, 1)
- Whether action $n$ is available $a_n$ (discrete, 2), for $n \in \{1, \ldots, 9\}$

From this raw feature set, we formed a large, sparse set of *flat* features suitable for learning with linear function approximation. First, to capture interactions, we considered all pairwise combinations of the $9$ actions, of which there are $36$ ($= \frac{9 \times (9-1)}{2}$). Next, every pair of actions has four possible values based on their availability (both available, $2 \times$ one available and the second not, both unavailable). Finally, each action pair-availability setting has three scalar features: for the availability that describes the action pair, these three features are set to $[p_c, p_t, 1]^\top$, and for the others $[0, 0, 0]^\top$. Altogether this yields $432$ ($= 36 \times 4 \times 3$) *untuned flat* features per action type.

Experiments were conducted by partitioning the synthetic corpus into $10$ subsets of $2000$ dialogs. We process each of these $10$ subsets independently and will refer to them as trials. For each trial, we trained three RL dialog managers: the RL baseline with the hand-tuned raw feature set, the RL baseline with untuned raw set, and LSPI-FFS with the untuned flat feature set. (The clustering done by the RL baseline automatically considers interactions between raw features.) For each RL method, we obtained the resulting dialog manager after the first $N$ dialogs of training, where $N$ was varied from $10$ to $2000$. The policy learned after $N$ dialogs was then run with the simulated user for $1000$ dialogs. The results of the $10$ trials were then averaged together. For LSPI-FFS we used $k' = 80$, i.e. the $80$ most useful features per action. The hand-crafted dialog manager (with no RL) was also run for $1000$ simulated dialogs.

Results are shown in Figure Fig. 1(a). The two curves for the baseline RL method converge to the same task completion ratio, indicating the (larger) untuned feature set contains sufficient information for learning a good policy. LSPI-FFS performs almost identically to the baseline RL method on the untuned feature set, yet is using a small fraction of the features ($80$ of the $432$ flat features per action). This implies that it is performing feature selection effectively. We also tried running (unmodified) LSPI and found it was computationally infeasible with the $432$ flat features per action. In other words, our method enables LSPI to scale to a real-world RL dialog problem.

We next augmented the untuned feature set by adding 1 or 2

[1] If user satisfaction data were available to us, there are ways of inferring a reward function that maximizes expected user satisfaction [13, 14]. However our data was collected from real callers (not usability subjects), for which we don't have user satisfaction data. So our reward function is handcrafted based on our goals for the system.

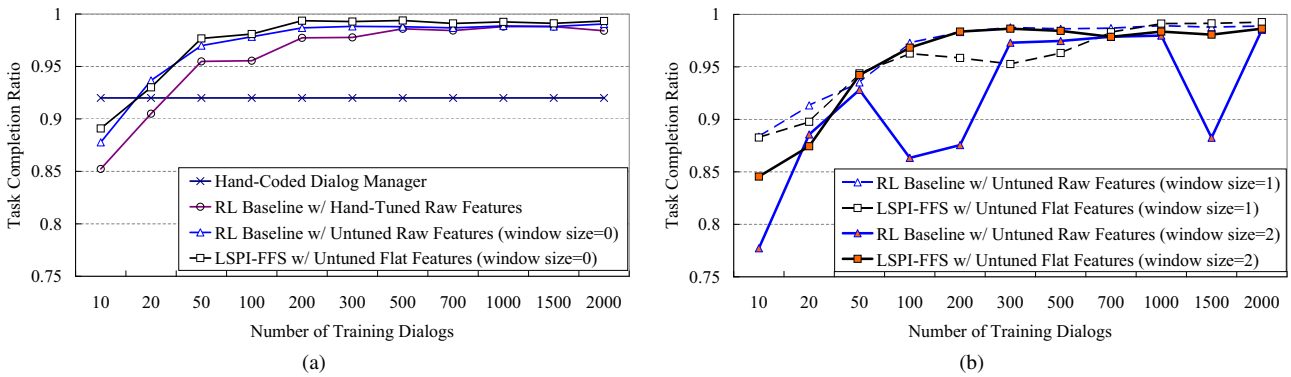


Figure 1: Average task completion ratios of policies learned by various algorithms. Average total rewards and average interaction lengths had qualitatively similar curves. See text for details.

timesteps of history, increasing the number of raw features from 11 to 22 for a history window of 1 timestep, and 33 for a history window of 2 timesteps. The untuned flat feature set grew from 432 features per action to 3060 and 9828 per action for 1 and 2 timesteps of history, respectively. We followed the same training procedure as above using the untuned feature set with the baseline RL and LSPI-FFS methods. For these experiments for LSPI-FFS we used $k' = 100$, *i.e.*, the 100 most useful features per action. Results are shown in Figure Fig. 1(b).

With one timestep of history, the baseline RL and LSPI-FFS methods both appear to converge after about 1000 dialogs; however with a window size of 2, the baseline RL method shows clear signs of instability (dips at $100 - 200$, and 1500 training dialogs), whereas LSPI-FFS does not. On inspection, we found that these dips were caused when one of the 10 baseline trials yielded a very poor policy. Thus these experiments illustrate the key distinction of our approach: whereas the baseline RL method struggles to make use of a large number of features (sometimes failing to find a good policy), LSPI-FFS appears quite stable and converges consistently, implying that it can effectively and automatically prune the large set down to a small, useful subset.

Past work has suggested history information can improve performance [15]. Although we did not observe that trend in Figure 1, this may be because some of our features already include history information (the probability that the top callee and top phone type are correct are posteriors over the whole dialog), or because our task contains a relatively small number (2) of slots. We hope to study this more when we apply LSPI-FFS to larger tasks.

## 5. Conclusions

In this work, we have investigated a method for scaling the number of features used by a reinforcement-learning based spoken dialog system. A comparison to a baseline RL method used to build a real-world dialog system shows that it is better able to handle large numbers of features, evaluated in dialog simulation. Our method extends least-squares policy iteration, which has a variety of attractive properties for dialog systems.

With this method, a designer can propose a large set of potentially useful features and side-step the difficult task of manually hunting for the useful subset. We hope this method will remove some of the "art" from deploying reinforcement-learning dialog systems, moving this technology a step closer to commercial readiness.

## 6. References


[1] E. Levin, R. Pieraccini, and W. Eckert, "A stochastic model of human-machine interaction for learning dialog strategies," *IEEE Trans. Speech Audio Process.*, vol. 8, no. 1, pp. 11–23, 2000.

[2] N. Roy, J. Pineau, and S. Thrun, "Spoken dialogue management using probabilistic reasoning," in *ACL*, 2000, pp. 93–100.

[3] J. Henderson, O. Lemon, and K. Georgila, "Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets," *Computational Linguistics*, vol. 34, no. 4, pp. 487–511, 2008.

[4] B. Thomson, J. Schatzmann, and S. Young, "Bayesian update of dialogue state for robust dialogue systems," in *ICASSP*, 2008.

[5] J. D. Williams, "Integrating expert knowledge into POMDP optimization for spoken dialog systems," in *AAAI-08 Workshop on Advancements in POMDP Solvers*, 2008.

[6] R. Parr, C. Painter-Wakefield, L. Li, and M. L. Littman, "Analyzing feature generation for value-function approximation," in *Int. Conf. Mach. Learning*, 2007, pp. 737–744.

[7] S. Mahadevan and M. Maggioni, "Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes," *Journall of Machine Learning Research*, vol. 8, pp. 2169–2231, 2007.

[8] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journall of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

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

[10] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[11] J. D. Williams, "Demonstration of a POMDP voice dialer," In ACL/HLT, 2008.

[12] ——, "The best of both worlds: Unifying conventional dialog systems and pomdps," in *ICSLP-08*, 2008.

[13] M. Walker, "An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email," *Journal of Articial Intelligence Research*, vol. 12, pp. 387–416, 2000.

[14] V. Rieser and O. Lemon, "Automatic learning and evaluation of user-centered objective functions for dialogue system optimisation," in *Proc LREC, Marrakech*, 2008.

[15] M. Frampton and O. Lemon, "Using dialogue acts to learn better repair strategies for spoken dialogue systems," in *Proc ICASSP, Las Vegas*, 2008.