Sparse Approximate Dynamic Programming for Dialog Management

Senthilkumar Chandramohan, Matthieu Geist, Olivier Pietquin
SUPELEC - IMS Research Group, Metz - France.
{senthilkumar.chandramohan, matthieu.geist, olivier.pietquin}@supelec.fr

Abstract

Spoken dialogue management strategy optimization by means of Reinforcement Learning (RL) is now part of the state of the art. Yet, there is still a clear mismatch between the complexity implied by the required naturalness of dialogue systems and the inability of standard RL algorithms to scale up. Another issue is the sparsity of the data available for training in the dialogue domain which can not ensure convergence of most of RL algorithms. In this paper, we propose to combine a sample-efficient generalization framework for RL with a feature selection algorithm for the learning of an optimal spoken dialogue management strategy.

1 Introduction

Optimization of dialogue management strategies by means of Reinforcement Learning (RL) (Sutton and Barto, 1998) is now part of the state of the art in the research area of Spoken Dialogue Systems (SDS) (Levin and Pieraccini, 1998; Singh et al., 1999; Pietquin and Dutoit, 2006; Williams and Young, 2007). It consists in casting the dialogue management problem into the Markov Decision Processes (MDP) paradigm (Bellman, 1957) and solving the associated optimization problem. Yet, there is still a clear mismatch between the complexity implied by the required naturalness of the dialogue systems and the inability of standard RL algorithms to scale up. Another issue is the sparsity of the data available for training in the dialogue domain because collecting and annotating data is very time consuming. Yet, RL algorithms are very data demanding and low amounts of data can not ensure convergence of most of RL algorithms. This latter problem has been extensively studied in the recent years and is addressed by simulating new dialogues thanks to a statistical model of human-machine interaction (Pietquin, 2005) and user modeling (Eckert et al., 1997; Pietquin and Dutoit, 2006; Schatzmann et al., 2006). However, this results in a variability of the learned strategy depending on the user modeling method (Schatzmann et al., 2005) and no common agreement exists on the best user model.

The former problem, that is dealing with complex dialogue systems within the RL framework, has received much less attention. Although some works can be found in the SDS literature it is far from taking advantage of the large amount of machine learning literature devoted to this problem. In (Williams and Young, 2005), the authors reduce the complexity of the problem (which is actually a Partially Observable MDP) by automatically condensing the continuous state space in a so-called *summary space*. This results in a clustering of the state space in a discrete set of states on which standard RL algorithms are applied. In (Henderson et al., 2008), the authors use a linear approximation scheme and apply the SARSA(λ) algorithm (Sutton and Barto, 1998) in a batch setting (from data and not from interactions or simulations). This algorithm was actually designed for online learning and is known to converge very slowly. It therefore requires a lot of data and especially in large state spaces. Moreover, the choice of the features used for the linear approximation is particularly simple since features are the state variables themselves. The approximated function can therefore not be more complex than an hyper-plane in the state variables space. This drawback is shared by the approach of (Li et al., 2009) where a batch algorithm (Least Square Policy Iteration or LSPI) is combined to a pruning method to only keep the most meaningful features. In addition the complexity of LSPI is $O(p^3)$.

In the machine learning community, this issue is actually addressed by function approximation accompanied with dimensionality reduction. The

Proceedings of SIGDIAL 2010: the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue, pages 107–115,
The University of Tokyo, September 24-25, 2010.

data sparsity problem is also widely addressed in this literature, and sample-efficiency is one main trend of research in this field. In this paper, we propose to combine a sample-efficient batch RL algorithm (namely the Fitted Value Iteration (FVI) algorithm) with a feature selection method in a novel manner and to apply this original combination to the learning of an optimal spoken dialogue strategy. Although the algorithm uses a linear combination of features (or basis functions), these features are much richer in their ability of representing complex functions.

The ultimate goal of this research is to provide a way of learning optimal dialogue policies for a large set of situations from a small and fixed set of annotated data in a tractable way.

The rest of this paper is structured as follows. Section 2 gives a formal insight of MDP and briefly reminds the casting of the dialogue problem into the MDP framework. Section 3.2 provides a description of approximate Dynamic Programming along with LSPI and FVI algorithms. Section 4 provides an overview on how LSPI and FVI can be combined with a feature selection scheme (which is employed to learn the representation of the Q-function from the dialogue corpus). Our experimental set-up, results and a comparison with state-of-the-art methods are presented in Section 5. Eventually, Section 6 concludes.

2 Markov Decision Processes

The MDP (Puterman, 1994) framework is used to describe and solve sequential decision making problems or equivalently optimal control problems in the case of stochastic dynamic systems. An MDP is formally a tuple $\{S, A, P, R, \gamma\}$ where S is the (finite) state space, A the (finite) action space, $P \in \mathcal{P}(S)^{S\times A}$ the family of Markovian transition probabilities[1], $R \in \mathbb{R}^{S\times A\times S}$ the reward function and γ the discounting factor ($0 \leq \gamma \leq 1$). According to this formalism, a system to be controlled steps from state to state ($s \in S$) according to transition probabilities P as a consequence of the controller's actions ($a \in A$). After each transition, the system generates an immediate reward (r) according to its reward function R. How the system is controlled is modeled with a so-called *policy* $\pi \in A^S$ mapping states to actions. The quality of a policy is quantified by the so-called value function which maps each state to the expected discounted cumulative reward given that the agent starts in this state and follows the policy π: $V^\pi(s) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, \pi]$. An optimal policy π^* maximizes this function for each state: $\pi^* = \text{argmax}_\pi V^\pi$. Suppose that we are given the optimal value function V^* (that is the value function associated to an optimal policy), deriving the associated policy would require to know the transition probabilities P. Yet, this is usually unknown. This is why the state-action value (or Q-) function is introduced. It adds a degree of freedom on the choice of the first action:

$$Q^\pi(s,a) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi] \quad (1)$$

The optimal policy is noted π^* and the related Q-function $Q^*(s,a)$. An action-selection strategy that is greedy according to this function ($\pi(s) = \text{argmax}_a Q^*(s,a)$) provides an optimal policy.

2.1 Dialogue as an MDP

The casting of the spoken dialogue management problem into the MDP framework (MDP-SDS) comes from the equivalence of this problem to a sequential decision making problem. Indeed, the role of the dialogue manager (or the decision maker) is to select and perform dialogue acts (actions in the MDP paradigm) when it reaches a given dialogue turn (state in the MDP paradigm) while interacting with a human user. There can be several types of system dialogue acts. For example, in the case of a restaurant information system, possible acts are *request(cuisine_type)*, *provide(address)*, *confirm(price_range)*, *close etc.* The dialogue state is usually represented efficiently by the Information State paradigm (Larsson and Traum, 2000). In this paradigm, the dialogue state contains a compact representation of the history of the dialogue in terms of system acts and its subsequent user responses (user acts). It summarizes the information exchanged between the user and the system until the considered state is reached.

A dialogue management strategy is thus a mapping between dialogue states and dialogue acts. Still following the MDP's definitions, the optimal strategy is the one that maximizes some cumulative function of rewards collected all along the interaction. A common choice for the immediate reward is the contribution of each action to user satisfaction (Singh et al., 1999). This subjective

[1]Notation $f \in A^B$ is equivalent to $f : B \to A$

reward is usually approximated by a linear combination of objective measures like dialogue duration, number of ASR errors, task completion *etc*. (Walker et al., 1997).

3 Solving MDPs

3.1 Dynamic Programming

Dynamic programming (DP) (Bellman, 1957) aims at computing the optimal policy π^* if the transition probabilities and the reward function are known.

First, the *policy iteration* algorithm computes the optimal policy in an iterative way. The initial policy is arbitrary set to π_0. At iteration k, the policy π_{k-1} is evaluated, that is the associated Q-function $Q^{\pi_{k-1}}(s,a)$ is computed. To do so, the Markovian property of the transition probabilities is used to rewrite Equation (1) as :

$$\begin{aligned} Q^{\pi}(s,a) &= E_{s'|s,a}[R(s,a,s') + \gamma Q^{\pi}(s',\pi(s'))] \\ &= T^{\pi}Q^{\pi}(s,a) \end{aligned} \quad (2)$$

This is the so-called Bellman evaluation equation and T^{π} is the Bellman evaluation operator. T^{π} is linear and therefore this defines a linear system that can be solved by standard methods or by an iterative method using the fact that Q^{π} is the unique fixed-point of the Bellman evaluation operator (T^{π} being a contraction): $\hat{Q}^{\pi}_i = T^{\pi}\hat{Q}^{\pi}_{i-1}, \quad \forall \hat{Q}^{\pi}_0 \ \lim_{i\to\infty} \hat{Q}^{\pi}_i = Q^{\pi}$. Then the policy is improved, that is π_k is greedy respectively to $Q^{\pi_{k-1}}$: $\pi_k(s) = \operatorname{argmax}_{a\in A} Q^{\pi_{k-1}}(s,a)$. Evaluation and improvement steps are iterated until convergence of π_k to π^* (which can be demonstrated to happen in a finite number of iterations when $\pi_k = \pi_{k-1}$).

The *value iteration* algorithm aims at estimating directly the optimal state-action value function Q^* which is the solution of the Bellman optimality equation (or equivalently the unique fixed-point of the Bellman optimality operator T^*):

$$\begin{aligned} Q^*(s,a) &= E_{s'|s,a}[R(s,a,s') + \gamma \max_{b\in A} Q^*(s',b)] \\ &= T^*Q^*(s,a) \end{aligned} \quad (3)$$

The T^* operator is not linear, therefore computing Q^* via standard system-solving methods is not possible. However, it can be shown that T^* is also a contraction (Puterman, 1994). Therefore, according to Banach fixed-point theorem, Q^* can be estimated using the following iterative way:

$$\hat{Q}^*_i = T^*\hat{Q}^*_{i-1}, \quad \forall \hat{Q}^*_0 \ \lim_{i\to\infty} \hat{Q}^*_i = Q^* \quad (4)$$

However, the convergence takes an infinite number of iterations. Practically speaking, iterations are stopped when some criterion is met, classically a small difference between two iterations: $\|\hat{Q}^*_i - \hat{Q}^*_{i-1}\| < \xi$. The estimated optimal policy (which is what we are ultimately interested in) is greedy respectively to the estimated optimal Q-function: $\hat{\pi}^*(s) = \operatorname{argmax}_{a\in A} \hat{Q}^*(s,a)$.

3.2 Approximate Dynamic Programming

DP-based approaches have two drawbacks. First, they assume the transition probabilities and the reward function to be known. Practically, it is rarely true and especially in the case of spoken dialogue systems. Most often, only examples of dialogues are available which are actually trajectories in the state-action space. Second, it assumes that the Q-function can be exactly represented. However, in real world dialogue management problems, state and action spaces are often too large (even continuous) for such an assumption to hold. Approximate Dynamic Programming (ADP) aims at estimating the optimal policy from trajectories when the state space is too large for a tabular representation. It assumes that the Q-function can be approximated by some parameterized function $\hat{Q}_\theta(s,a)$. In this paper, a linear approximation of the Q-function will be assumed: $\hat{Q}_\theta(s,a) = \theta^T\phi(s,a)$. where $\theta \in \mathbb{R}^p$ is the parameter vector and $\phi(s,a)$ is the set of p basis functions. All functions expressed in this way define a so-called *hypothesis space* $\mathcal{H} = \{\hat{Q}_\theta | \theta \in \mathbb{R}^p\}$. Any function Q can be *projected* onto this hypothesis space by the operator Π defined as

$$\Pi Q = \operatorname*{argmin}_{\hat{Q}_\theta \in \mathcal{H}} \|Q - \hat{Q}_\theta\|^2. \quad (5)$$

The goal of the ADP algorithms explained in the subsequent sections is to compute the best set of parameters θ given the basis functions.

3.2.1 Least-Squares Policy Iteration

The least-squares policy iteration (LSPI) algorithm has been introduced by Lagoudakis and Parr (2003). The underlying idea is exactly the same as for policy iteration: interleaving evaluation and improvement steps. The improvement steps are same as before, but the evaluation step should learn an approximate representation of the Q-function using samples. In LSPI, this is done using the Least-Squares Temporal Differences (LSTD) algorithm of Bradtke and Barto (1996).

LSTD aims at minimizing the distance between the approximated Q-function $\hat{Q}_\theta$ and the projection onto the hypothesis space of its image through the Bellman evaluation operator $\Pi T^\pi \hat{Q}_\theta$: $\theta_\pi = \text{argmin}_{\theta \in \mathbb{R}^p} \|\hat{Q}_\theta - \Pi T^\pi \hat{Q}_\theta\|^2$. This can be interpreted as trying to minimize the difference between the two sides of the Bellman equation (1) (which should ideally be zero) in the hypothesis space. Because of the approximation, this difference is most likely to be non-zero.

Practically, T^π is not known, but a set of N transitions $\{(s_j, a_j, r_j, s'_j)_{1 \leq j \leq N}\}$ is available. LSTD therefore solves the following optimization problem: $\theta_\pi = \text{argmin}_\theta \sum_{j=1}^N C_j^N(\theta)$ where $C_j^N(\theta) = (r_j + \gamma \hat{Q}_{\theta_\pi}(s'_j, \pi(s'_j)) - \gamma \hat{Q}_\theta(s_j, a_j))^2$. Notice that θ_π appears in both sides of the equation, which renders this problem difficult to solve. However, thanks to the linear parametrization, it admits an analytical solution, which defines the LSTD algorithm:

$$\theta_\pi = (\sum_{j=1}^N \phi_j \Delta\phi_j^\pi)^{-1} \sum_{j=1}^N \phi_j r_j \qquad (6)$$

with $\phi_j = \phi(s_j, a_j)$ and $\Delta\phi_j^\pi = \phi(s_j, a_j) - \gamma\phi(s'_j, \pi(s'_j))$.

LSPI is initialized with a policy π_0. Then, at iteration k, the Q-function of policy π_{k-1} is estimated using LSTD, and π_k is greedy respectively to this estimated state-action value function. Iterations are stopped when some stopping criterion is met (*e.g.*, small differences between consecutive policies or associated Q-functions).

3.2.2 Least-Squares Fitted Value Iteration

The Fitted Value Iteration (FVI) class of algorithms (Bellman and Dreyfus, 1959; Gordon, 1995; Ernst et al., 2005) generalizes value iteration to model-free and large state space problems. The T^* operator (eq. (3)) being a contraction, a straightforward idea would be to apply it iteratively to the approximation similarly to eq. (4): $\hat{Q}_{\theta_k} = T^* \hat{Q}_{\theta_{k-1}}$. However, $T^* \hat{Q}_\theta$ does not necessarily lie in $\mathcal{H}$, it should thus be projected again onto the hypothesis space $\mathcal{H}$. By considering the same projection operator Π as before, this leads to finding the parameter vector θ satisfying: $\hat{Q}_\theta^* = \Pi T^* \hat{Q}_\theta^*$. The fitted-$Q$ algorithm (a special case of FVI) assumes that the composed ΠT^* operator is a contraction and therefore admits an unique fixed point, which is searched for through the classic iterative scheme: $\hat{Q}_{\theta_k} = \Pi T^* \hat{Q}_{\theta_{k-1}}$. However, the model (transition probabilities and the reward function) is usually not known, therefore a *sampled* Bellman optimality operator $\hat{T}^*$ is considered instead. For a transition sample (s_j, a_j, r_j, s'_j), it is defined as: $\hat{T}^* Q(s_j, a_j) = r_j + \gamma \max_{a \in A} Q(s'_j, a)$. This defines the general fitted-Q algorithm (θ_0 being chosen by the user): $\hat{Q}_{\theta_k} = \Pi \hat{T}^* \hat{Q}_{\theta_{k-1}}$. Fitted-$Q$ can then be specialized by choosing how $\hat{T}^* \hat{Q}_{\theta_{k-1}}$ is projected onto the hypothesis space, that is the supervised learning algorithm that solves the projection problem of eq. (5). The least squares algorithm is chosen here.

The parametrization being linear, and a training base $\{(s_j, a_j, r_j, s'_j)_{1 \leq j \leq N}\}$ being available, the least-squares fitted-Q (LSFQ for short) is derived as follows (we note $\phi(s_j, a_j) = \phi_j$):

$$\theta_k = \underset{\theta \in \mathbb{R}^p}{\text{argmin}} \sum_{j=1}^N (\hat{T}^* \hat{Q}_{\theta_{k-1}}(s_j, a_j) - \hat{Q}_\theta(s_j, a_j))^2 \qquad (7)$$
$$= (\sum_{j=1}^N \phi_j \phi_j^T)^{-1} \sum_{j=1}^N \phi_j (r_j + \gamma \max_{a \in A} (\theta_{k-1}^T \phi(s'_j, a)))$$

Equation (7) defines an iteration of the proposed linear least-squares-based fitted-Q algorithm. An initial parameter vector θ_0 should be chosen, and iterations are stopped when some criterion is met (maximum number of iterations or small difference between two consecutive parameter vector estimates). Assuming that there are M iterations, the optimal policy is estimated as $\hat{\pi}^*(s) = \text{argmax}_{a \in A} \hat{Q}_{\theta_M}(s, a)$.

4 Learning a sparse parametrization

LSPI and LSFQ (FVI) assume that the basis functions are chosen beforehand. However, this is difficult and problem-dependent. Thus, we propose to combine these algorithms with a scheme which *learns* the representation from dialogue corpora.

Let's place ourselves in a general context. We want to learn a parametric representation for an approximated function $f_\theta(z) = \theta^T \phi(z)$ from samples $\{z_1, \ldots, z_N\}$. A classical choice is to choose a kernel-based representation (Scholkopf and Smola, 2001). Formally, a kernel $K(z, \tilde{z}_i)$ is a continuous, positive and semi-definite function (*e.g.*, Gaussian or polynomial kernels) centered on $\tilde{z}_i$. The feature vector $\phi(z)$ is therefore of the form: $\phi(z) = \begin{pmatrix} K(z, \tilde{z}_1) & \ldots & K(z, \tilde{z}_p) \end{pmatrix}$. The question this section answers is the following: given the training basis $\{z_1, \ldots, z_N\}$ and a kernel

K, how to choose the number p of basis functions and the associated kernel centers $(\tilde{z}_1, \ldots, \tilde{z}_p)$?

An important result about kernels is the Mercer theorem, which states that for each kernel K there exists a mapping $\varphi : z \in Z \rightarrow \varphi(z) \in \mathcal{F}$ such that $\forall z_1, z_2 \in Z$, $K(z_1, z_2) = \langle \varphi(z_1), \varphi(z_2) \rangle$ (in short, K defines a dot product in $\mathcal{F}$). The space $\mathcal{F}$ is called the feature space, and it can be of infinite dimension (*e.g.*, Gaussian kernel), therefore φ cannot always be explicitly built. Given this result and from the bilinearity of the dot product, f_θ can be rewritten as follows: $f_\theta(z) = \sum_{i=1}^{p} \theta_i K(z, \tilde{z}_i) = \langle \varphi(z), \sum_{i=1}^{p} \theta_i \varphi(\tilde{z}_i) \rangle$. Therefore, a kernel-based parametrization corresponds to a linear approximation in the feature space, the weight vector being $\sum_{i=1}^{p} \theta_i \varphi(\tilde{z}_i)$. This is called the *kernel trick*. Consequently, kernel centers $(\tilde{z}_1, \ldots, \tilde{z}_p)$ should be chosen such that $(\varphi(\tilde{z}_1), \ldots, \varphi(\tilde{z}_p))$ are linearly independent in order to avoid using redundant basis functions. Moreover, kernel centers should be chosen among the training samples. To sum up, learning such a parametrization reduces to finding a dictionary $\mathcal{D} = (\tilde{z}_1, \ldots, \tilde{z}_p) \in \{z_1, \ldots, z_N\}$ such that $(\varphi(\tilde{z}_1), \ldots, \varphi(\tilde{z}_p))$ are linearly independent and such that they span the same subspace as $(\varphi(z_1), \ldots, \varphi(z_N))$. Engel et al. (2004) provides a dictionary method to solve this problem, briefly sketched here.

The training base is sequentially processed, and the dictionary is initiated with the first sample: $\mathcal{D}_1 = \{z_1\}$. At iteration k, a dictionary $\mathcal{D}_{k-1}$ computed from $\{z_1, \ldots, z_{k-1}\}$ is available and the k^{th} sample z_k is considered. If $\varphi(z_k)$ is linearly independent of $\varphi(\mathcal{D}_{k-1})$, then it is added to the dictionary: $\mathcal{D}_k = \mathcal{D}_{k-1} \cup \{z_k\}$. Otherwise, the dictionary remains unchanged: $\mathcal{D}_k = \mathcal{D}_{k-1}$. Linear dependency can be checked by solving the following optimization problem (p_{k-1} being the size of $\mathcal{D}_{k-1}$): $\delta = \text{argmin}_{w \in \mathbb{R}^{p_{k-1}}} \|\varphi(z_k) - \sum_{i=1}^{p_{k-1}} w_i \varphi(\tilde{z}_i)\|^2$. Thanks to the kernel trick (that is the fact that $\langle \varphi(z_k), \varphi(\tilde{z}_i) \rangle = K(z_k, \tilde{z}_i)$) and to the bilinearity of the dot product, this optimization problem can be solved analytically and without computing explicitly φ. Formally, linear dependency is satisfied if $\delta = 0$. However, an approximate linear dependency is allowed, and $\varphi(z_k)$ will be considered as linearly dependent of $\varphi(\mathcal{D}_{k-1})$ if $\delta < \nu$, where ν is the so-called *sparsification factor*. This allows controlling the trade-off between quality of the representation and its sparsity. See Engel et al. (2004) for details as well as an efficient implementation of this dictionary approach.

4.1 Resulting algorithms

We propose to combine LSPI and LSFQ with the sparsification approach exposed in the previous section: a kernel is chosen, the dictionary is computed and then LSPI or LSFQ is applied using the learnt basis functions. For LSPI, this scheme has been proposed before by Xu et al. (2007) (with the difference that they generate new trajectories at each iteration whereas we use the same for all iterations). The proposed sparse LSFQ algorithm is a novel contribution of this paper.

We start with the sparse LSFQ algorithm. In order to train the dictionary, the inputs are needed (state-action couples in this case), but not the outputs (reward are not used). For LSFQ, the input space remains the same over iterations, therefore the dictionary can be computed in a preprocessing step from $\{(s_j, a_j)_{1 \leq j \leq N}\}$. Notice that the matrix $(\sum_{j=1}^{N} \phi_j \phi_j^T)^{-1}$ remains also the same over iterations, therefore it can be computed in a preprocessing step too. The proposed sparse LSFQ algorithm is summarized in appendix Algorithm 1.

For the sparse LSPI algorithm, things are different. This time, the inputs depend on the iteration. More precisely, at iteration k, the input is composed of state-action couples (s_j, a_j) but also of transiting state-action couples $(s'_j, \pi_{k-1}(s'_j))$. Therefore the dictionary has to be computed at each iteration from $\{(s_j, a_j)_{1 \leq j \leq N}, (s'_j, \pi_{k-1}(s'_j))_{1 \leq j \leq N}\}$. This defines the parametrization which is considered for the Q-function evaluation. The rest of the algorithm is as for the classic LSPI and it is summarized in appendix Algorithm 2.

Notice that sparse LSFQ has a lower computational complexity than the sparse LSPI. For sparse LSFQ, dictionary and the matrix P^{-1} are computed in a preprocessing step, therefore the complexity per iteration is in $O(p^2)$, with p being the number of basis functions computed using the dictionary method. For LSPI, the inverse matrix depends on the iteration, as well as the dictionary, therefore the computational complexity is in $O(p_k^3)$ per iteration, where p_k is the size of the dictionary computed at the k^{th} iteration.

5 Experimental set-up and results

5.1 Dialogue task and RL parameters

The experimental setup is a form-filling dialogue system in the tourist information domain similar to the one studied in (Lemon et al., 2006). The system aims to give information about restaurants in the city based on specific user preferences. Three slots are considered: (i) location, (ii) cuisine and (iii) price-range of the restaurant. The dialogue state has three continuous components ranging from 0 to 1, each representing the average of filling and confirmation confidence of the corresponding slots. The MDP SDS has 13 actions: Ask-slot (3 actions), Explicit-confirm (3 actions), Implicit-confirm and Ask-slot value (6 actions) and Close-dialogue (1 action). The γ parameter was set to 0.95 in order to encourage delayed rewards and also to induce an implicit penalty for the length of the dialogue episode. The reward function R is presented as follows: every correct slot filling is awarded 25, every incorrect slot filling is awarded -75 and every empty slot filling is awarded -300. The reward is awarded at the end of the dialogue.

5.2 Dialogue corpora for policy optimization

So as to perform sparse LSFQ or sparse LSPI, a dialogue corpus which represents the problem space is needed. As for any batch learning method, the samples used for learning should be chosen (if they can be chosen) to span across the problem space. In this experiment, a user simulation technique was used to generate the data corpora. This way, the sensibility of the method to the size of the training data-set could be analyzed (available human-dialogue corpora are limited in size). The user simulator was plugged to the DIPPER (Lemon et al., 2006) dialogue management system to generate dialogue samples. To generate data, the dialogue manager strategy was jointly based on a simple hand-coded policy (which aims only to fill all the slots before closing the dialogue episode irrespective of slot confidence score *i.e.,*) and random action selection.

Randomly selected system acts are used with probability ϵ and hand-coded policy selected system acts are used with probability (1-ϵ). During our data generation process the ϵ value was set to 0.9. Rather than using a fully random policy we used an ϵ-greedy policy to ensure that the problem space is well sampled and in the same time at least few episodes have successful completion of task compared to a totally random policy. We ran 56,485 episodes between the policy learner and an unigram user simulation, using the ϵ-greedy policy (of which 65% are successful task completion episodes) and collected 393,896 dialogue turns (state transitions). The maximum episode length is set as 100 dialogue turns. The dialogue turns (samples) are then divided into eight different training sets each with 5.10^4 samples.

5.3 Linear representation of Q-function

Two different linear representations of the Q-function were used. First, a set of basis functions computed using the dictionary method outlined in Section 4 is used. A Gaussian kernel is used for the dictionary computation (σ = 0.25). The number of elements present in the dictionary varied based on the number of samples used for computation and the sparsification factor. It was observed during the experiments that including a constant term to the Q-function representation (value set to 1) in addition to features selected by the dictionary method avoided weight divergence. Our second representation of Q-function used a set of hand-picked features presented as a set of Gaussian functions, centered in μ_i and with the same standard deviation $\sigma_i = \sigma$). Our RBF network had 3 Gaussians for each dimension in the state vector and considering that we have 13 actions, in total we used 351 (*i.e,* $3^3 \times 13$) features for approximating the Q-function. This allows considering that each state variable contributes to the value function differently according to its value contrarily to similar work (Li et al., 2009; Henderson et al., 2008) that considers linear contribution of each state variable. Gaussians were centered at μ_i = 0.0, 0.5, 1.0 in every dimension with a standard deviation $\sigma_i = \sigma = 0.25$. Our stopping criteria was based on comparison between L_1 norm of succeeding weights and a threshold ξ which was set to 10^{-2} *i.e,* convergence if $\sum_i \left(|\theta_i^n - \theta_i^{n-1}|^1\right) < \xi$, where n is the iteration number. For sparse LSPI since the dictionary is computed during each iteration, stopping criteria based on ξ is not feasible thus the learning was stopped after 30 iterations.

5.4 Evaluation of learned policy

We ran a set of learning iterations using two different representations of Q-function and with different numbers of training samples (one sample is a dialogue turn, that is a state transition $\{s, a, r, s'\}$). The number of samples used for training ranged

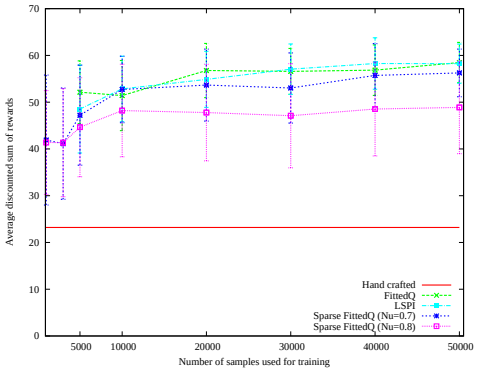

Figure 1: FittedQ policy evaluation statistics

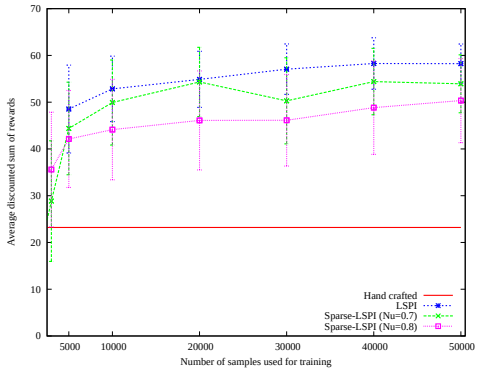

Figure 2: LSPI policy evaluation statistics

from 1.10^3 to 50.10^3 samples (no convergence of weights was observed with fewer samples than 1.10^3). The training is repeated for each of the 8 training data sets. Dictionary computed using different number of training samples and with ν=0.7 and 0.8 had a maximum of 367 and 306 elements respectively (with lower values of ν the number of features is higher than the hand-selected version). The policies learned were then tested using a unigram user simulation and the DIPPER dialogue management framework. Figures 1 and 2 show the average discounted sum of rewards of policies tested over 8×25 dialogue episodes.

5.5 Analysis of evaluation results

Our experimental results show that the dialogue policies learned using sparse SLFQ and LSPI with the two different Q-function representations perform significantly better than the hand-coded policy. Most importantly it can be observed from Figure 1 and 2 that the performance of sparse LSFQ and sparse LSPI (which uses the dictionary method for feature selection) are nearly as good as LSFQ and LSPI (which employs more numerous hand-selected basis functions). This shows the effectiveness of using the dictionary method for learning the representation of the Q-function from the dialogue corpora. For this specific problem the set of hand selected features seem to perform better than sparse LSPI and sparse LSFQ, but this may not be always the case. For complex dialogue management problems feature selection methods such as the one studied here will be handy since the option of manually selecting a good set of features will cease to exist.

Secondly it can be concluded that, similar to LSFQ and LSPI, the sparse LSFQ and sparse LSPI based dialogue management are also sample efficient and needs only few thousand samples (recall that a sample is a dialogue turn and not a dialogue episode) to learn fairly good policies, thus exhibiting a possibility to learn a good policy directly from very limited amount of dialogue examples. We believe this is a significant improvement when compared to the corpora requirement for dialogue management using other RL algorithms such as SARSA. However, sparse LSPI seems to result in poorer performance compared to sparse LSFQ.

One key advantage of using the dictionary method is that only mandatory basis functions are selected to be part of the dictionary. This results in fewer feature weights ensuring faster convergence during training. From Figure 1 it can also be observed that the performance of both LSFQ and LSPI (using hand selected features) are nearly identical. From a computational complexity point of view, LSFQ and LSPI roughly need the same number of iterations before the stopping criterion is met. However, reminding that the proposed LSFQ complexity is $O(p)^2$ per iteration whereas LSPI complexity is $O(p^3)$ per iteration, LSFQ is computationally less intensive.

6 Discussion and Conclusion

In this paper, we proposed two sample-efficient generalization techniques to learn optimal dialogue policies from limited amounts of dialogue examples (namely sparse LSFQ and LSPI). Particularly, a novel sparse LSFQ method has been proposed and was demonstrated to out-perform handcrafted and LSPI-based policies while using a limited number of features. By using a kernel-based approximation scheme, the power of representation of the state-action value function (or Q-function) is increased with comparison to state-of-

the-art algorithms (such as (Li et al., 2009; Henderson et al., 2008)). Yet the number of features is also increased. Using a sparsification algorithm, this number is reduced while policy performances are kept. In the future, more compact representation of the state-action value function will be investigated such as neural networks.

Acknowledgments

The work presented here is part of an ongoing research for CLASSiC project (Grant No. 216594, www.classic-project.org) funded by the European Commission's 7th Framework Programme (FP7).

References

Richard Bellman and Stuart Dreyfus. 1959. Functional approximation and dynamic programming. *Mathematical Tables and Other Aids to Computation*, 13:247–251.

Richard Bellman. 1957. *Dynamic Programming*. Dover Publications, sixth edition.

Steven J. Bradtke and Andrew G. Barto. 1996. Linear Least-Squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57.

Wieland Eckert, Esther Levin, and Roberto Pieraccini. 1997. User Modeling for Spoken Dialogue System Evaluation. In *ASRU'97*, pages 80–87.

Yaakov Engel, Shie Mannor, and Ron Meir. 2004. The Kernel Recursive Least Squares Algorithm. *IEEE Transactions on Signal Processing*, 52:2275–2285.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. 2005. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, 6:503–556.

Geoffrey Gordon. 1995. Stable Function Approximation in Dynamic Programming. In *ICML'95*.

James Henderson, Oliver Lemon, and Kallirroi Georgila. 2008. Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets. *Computational Linguistics, vol. 34(4), pp 487-511*.

Michail G. Lagoudakis and Ronald Parr. 2003. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149.

Staffan Larsson and David R. Traum. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering, vol. 6, pp 323–340*.

Oliver Lemon, Kallirroi Georgila, James Henderson, and Matthew Stuttle. 2006. An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system. In *EACL'06*, Morristown, NJ, USA.

Esther Levin and Roberto Pieraccini. 1998. Using markov decision process for learning dialogue strategies. In *ICASSP'98*.

Lihong Li, Suhrid Balakrishnan, and Jason Williams. 2009. Reinforcement Learning for Dialog Management using Least-Squares Policy Iteration and Fast Feature Selection. In *InterSpeech'09*, Brighton (UK).

Olivier Pietquin and Thierry Dutoit. 2006. A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Audio, Speech & Language Processing, 14(2): 589-599*.

Olivier Pietquin. 2005. A probabilistic description of man-machine spoken communication. In *ICME'05*, pages 410–413, Amsterdam (The Netherlands), July.

Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, April.

Jost Schatzmann, Matthew N. Stuttle, Karl Weilhammer, and Steve Young. 2005. Effects of the user model on simulation-based learning of dialogue strategies. In *ASRU'05*, December.

Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Engineering Review, vol. 21(2), pp. 97–126*.

Bernhard Scholkopf and Alexander J. Smola. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

Satinder Singh, Michael Kearns, Diane Litman, and Marilyn Walker. 1999. Reinforcement learning for spoken dialogue systems. In *NIPS'99*. Springer.

Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 3rd edition, March.

Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. 1997. PARADISE: A framework for evaluating spoken dialogue agents. In *ACL'97*, pages 271–280, Madrid (Spain).

Jason Williams and Steve Young. 2005. Scaling up pomdps for dialogue management: the summary pomdp method. In *ASRU'05*.

Jason D. Williams and Steve Young. 2007. Partially observable markov decision processes for spoken dialog systems. *Computer Speech and Language, vol. 21(2), pp. 393–422*.

Xin Xu, Dewen Hu, and Xicheng Lu. 2007. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, July.

Appendix

This appendix provides pseudo code for the algorithms described in the paper.

Algorithm 1: Sparse LSFQ.

Initialization;
Initialize vector θ_0, choose a kernel K and a sparsification factor ν;
Compute the dictionary;
$\mathcal{D} = \{(\tilde{s}_j, \tilde{a}_j)_{1 \leq j \leq p}\}$ from $\{(s_j, a_j)_{1 \leq j \leq N}\}$;
Define the parametrization;
$Q_\theta(s,a) = \theta^T \phi(s,a)$ with $\phi(s,a) = (K((s,a),(\tilde{s}_1,\tilde{a}_1)), \ldots, K((s,a),(\tilde{s}_p,\tilde{a}_p)))^T$;
Compute P^{-1};
$P^{-1} = (\sum_{j=1}^N \phi_j \phi_j^T)^{-1}$;
for $k = 1, 2, \ldots, M$ **do**
  Compute θ_k, see Eq. (7);
end
$\hat{\pi}_M^*(s) = \text{argmax}_{a \in A} \hat{Q}_{\theta_M}(s,a)$;

Algorithm 2: Sparse LSPI.

Initialization;
Initialize policy π_0, choose a kernel K and a sparsification factor ν;
for $k = 1, 2, \ldots$ **do**
  Compute the dictionary;
  $\mathcal{D} = \{(\tilde{s}_j, \tilde{a}_j)_{1 \leq j \leq p_k}\}$ from $\{(s_j, a_j)_{1 \leq j \leq N}, (s'_j, \pi_{k-1}(s'_j))_{1 \leq j \leq N}\}$;
  Define the parametrization;
  $Q_\theta(s,a) = \theta^T \phi(s,a)$ with $\phi(s,a) = (K((s,a),(\tilde{s}_1,\tilde{a}_1)), \ldots, K((s,a),(\tilde{s}_{p_k},\tilde{a}_{p_k})))^T$;
  Compute θ_{k-1}, see Eq. (6);
  Compute π_k;
  $\pi_k(s) = \text{argmax}_{a \in A} \hat{Q}_{\theta_{k-1}}(s,a)$;
end