UNIVERSITY OF CAMBRIDGE
Department of Engineering

Gaussian processes for POMDP-based dialogue manager optimisation

Milica Gašić mg436@eng.cam.ac.uk
Steve Young sjy@eng.cam.ac.uk

Technical Report
CUED/F-INFENG/TR.684

18th January 2013

Department of Engineering
University of Cambridge
Trumpington Street
Cambridge, CB2 1PZ, U.K.

Abstract

A partially observable Markov decision process (POMDP) has been proposed as a dialogue model that enables automatic optimisation of the dialogue policy and provides robustness to speech understanding errors. Various approximations allow such a model to be used for building real-world dialogue systems. However, they require a large number of dialogues to train the dialogue policy and hence they typically rely on the availability of a user simulator. They also require significant designer effort to hand-craft the policy representation. We investigate the use of Gaussian processes (GPs) in policy modelling to overcome these problems. We show that GP policy optimisation can be implemented for a real world POMDP dialogue manager, and in particular: 1) we examine different formulations of a GP policy to minimise variability in the learning process; 2) we find that the use of GP increases the learning rate by an order of magnitude thereby allowing learning by direct interaction with human users; and 3) we demonstrate that designer effort can be substantially reduced by basing the policy directly on the full belief space thereby avoiding ad hoc feature space modelling. Overall, the GP approach represents an important step forward towards fully automatic dialogue policy optimisation in real world systems.

1 Introduction

Spoken dialogue systems enable human-computer interaction where the primary input is speech. As such they have innumerable benefits. However, building such systems to operate robustly is challenging as they are very sensitive to speech recognition errors and require a significant effort from the dialogue designer to define their behaviour. The focus of this paper is automatic optimisation of dialogue systems using a small amount of training data and minimal effort from the designer.

One way of describing human-computer dialogue is to view it as a sequence of dialogue turns in which each turn consists of a system utterance followed by a user utterance. In the traditional approach to building spoken dialogue systems, the system utterances are determined by hand-coded rules and depend on the input that is received from the speech recogniser [1, 2]. For a real-world dialogue system the number of such rules can be very large and a designer is required to manually implement each of them. Moreover, each rule requires an explicit error handler as otherwise a mis-recognition will be treated as a true user input. Finally, once deployed, such systems stay fixed unless the designer manually changes their behaviour.

A statistical approach to spoken dialogue systems has been proposed to automate learning of dialogue rules, increase robustness to speech recognition errors and enable adaptation to different users [3]. The Markov decision process (MDP) emerged as a model of dialogue which can automate learning of dialogue rules [4, 5, 6]. Here, the dialogue is modelled as a sequence of states s_t and in each state the system takes an action a_t, which corresponds to the system utterance. The state s_t is designed in such a way that it encapsulates sufficient information about the dialogue turns up to time t so that the model satisfies the Markov property. More specifically, the transition to any state depends only on the previous state and the action that was taken and this is modelled by a *transition probability* $P(s_{t+1}|s_t, a_t)$. In each state, the system receives a reward r_t, which is a measure of how good the system action is for that particular state. The aim is then to find a policy π which maps each state $s \in \mathcal{S}$ into an action $a \in \mathcal{A}$, $\pi(s) = a$, such that the total accumulated reward at the end of the dialogue is maximised. This is measured by the Q-function, which for every dialogue state s, system action a and policy π defines the expected discounted reward that can be obtained when action a is taken in state s and from then on policy π is followed

$$Q^\pi(s,a) = E_\pi\left(\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s, a_t = a\right), \tag{1}$$

where γ is a discount factor, $\gamma \in (0, 1]$, which favours good actions being taken sooner rather than later.

The reward function is typically defined to encourage short dialogues which provide all of the information requested by the user. The MDP model allows the optimal policy to be found automatically. This can be performed *off-line* if the transition probability and reward function are known—model-based reinforcement learning. Alternatively, the policy can be optimised *on-line* in direct interaction with the user—model-free reinforcement learning [7]. The latter approach has been successfully implemented for training a dialogue policy in interaction with a simulated user [4, 5, 6, 8, 9, 10, 11, 12]. However, similar to the rule-based approach, the MDP model assumes that the dialogue state correctly represents the user input and hence it must incorporate an error recovery strategy at every dialogue state.

A partially observable Markov decision process (POMDP) has been proposed as a more accurate model for dialogue which intrinsically deals with uncertainty from the recogniser. It thus has the potential to provide more robust operation [3, 13, 14, 15, 16]. It assumes that the dialogue state is only partially observable and depends on a noisy observation o_t

defined by an *observation probability* $P(o_t|s_t)$. Since the dialogue state is unobservable, at every dialogue step a distribution over all states is maintained, which is called the *belief state* b_t. It takes values $\mathbf{b} \in \mathcal{B}$, where $\mathcal{B}$ is a continuous space of dimensionality $|\mathcal{S}|$, namely $[0,1]^{|\mathcal{S}|}$. The dialogue policy then maps the belief state into an appropriate action at every dialogue turn, $\pi(\mathbf{b}) = a$. The process of exact belief state updating requires a summation over all states, which rapidly becomes intractable for very large state spaces:

$$\begin{aligned} b(s_{t+1} = s') \quad & \propto P(o_{t+1} = o'|s_{t+1} = s') \\ & \sum_{s \in \mathcal{S}} P(s_{t+1} = s'|a_t = a, s_t = s) b(s_t = s). \end{aligned} \tag{2}$$

However, there exist real-world dialogue systems based on the POMDP model which maintain an approximate belief state distribution in real-time throughout the dialogue. These include the Hidden Information State (HIS) system [17] and the Bayesian Update of Dialogue State (BUDS) system [16]. Policy optimisation, however, involves a large amount of training data and requires the designer to explicitly define the policy representation [18, 19]. This paper explores ways of overcoming these limitations.

2 POMDP policy optimisation

Exact policy optimisation for POMDPs is intractable for everything but very simple cases [20]. Approximate POMDP solutions, as in the point-based value iteration algorithm [21, 22], are only suitable for relatively small state/action problems and are thus not directly applicable to applications with large state spaces. Moreover, it is assumed that the observation space is discrete and finite which is not normally the case for real-world dialogue tasks. However, a POMDP with discrete state and observation spaces can be viewed as an MDP with a continuous state space [20] where the transition model of the MDP is defined by the belief update formula (2). Thus if an approximation of the POMDP belief state can be tractably maintained at every dialogue step, the POMDP can be approximated as a continuous state MDP. This allows standard MDP algorithms to be used for policy optimisation and this usually introduces the need for some form of parametrisation.

For example, it is proposed in [23] that the Q-function is parametrised as a linear combination of features from the belief state. Optimisation is then performed using a modified Sarsa algorithm. In the BUDS system, the policy is parametrised as a softmax function of features of the belief state [24]. Then, gradient methods and the natural-actor critic algorithm are used for policy optimisation. While this achieves tractability, in the case of the BUDS system it requires from 10^5 to 10^6 dialogues to train a policy, and this is only possible in interaction with a simulated user [19]. Moreover, the basis feature functions must be chosen by the designer and the solution is only optimal within the chosen basis. In [25], an algorithm is proposed that automatically selects useful features from an initial set of features predefined by a designer. An alternative method described in [26] automatically summarises information from the spoken language understanding component and then builds a POMDP using this summarised space to ensure tractability in learning. However, this still requires heuristics and does not facilitate the incorporation of prior knowledge regarding the task domain and user behaviour. A more principled approach is taken in [27] where Krylov iteration for lossless compression is used to compress the dialogue states, however the effectiveness of this approach in practice greatly depends on the definition of the reward function.

A more recent thread of research has focused on parametric models of the Q-function where the uncertainty of the approximation can be encoded in the estimate of the parameters. Here, a Gaussian distribution is placed over each parameter and Kalman filters are used for optimisation. It is shown that this leads to increased speed of training, however the designer is still required to select a set of basis functions [28].

In the HIS system [17], the belief space is heuristically mapped into a much smaller summary space, which is then discretised into a grid, allowing MDP reinforcement learning algorithms for discrete spaces to be used for the policy optimisation. In this case, the speed of convergence, together with the quality of the approximation depend on the number of grid-points. If the number is too small, then approximating a belief state point with its closest grid point can lead to suboptimal results since two very different belief state points could be approximated by the same grid point. Conversely, if the number of grid-points is very large, policy optimisation becomes intractable. In the case of the HIS system, grid-based policy optimisation requires 10^5 dialogues, which is still too large for the policy to be optimised in direct interaction with humans and the forced use of a summary space again raises optimality questions.

Our aim is to reduce designer effort in defining a suitable policy representation and to speed up the process of policy optimisation to avoid reliance on user simulators. For this purpose, we propose non-parametric policy modelling which achieves operational efficiency by exploiting the similarities between different parts of the belief space. For instance, if the system action is reliably estimated in one belief state, that estimate should contribute to the unexplored parts of the belief space. In addition, it is not only important to estimate the system action for a given belief state, but also to provide a measure of how confident the system is about that estimate. This information is a useful indication of the extent to which different parts of the belief space have been explored during policy optimisation. A Gaussian process (GP) is proposed because it is a non-parametric model of Bayesian inference that has these desirable properties.

The next section introduces non-parametric policy modelling. It explains how the Q-function can be modelled as a Gaussian process and how a stochastic policy can be derived from this model that is well-suited for on-line learning. Then, in Section 4, experimental results using the BUDS dialogue manager are presented. Several issues concerning policy design are examined with the aim of reducing variability in the learning process. The resulting GP-based policy is shown to learn faster than a standard method and this is confirmed in a human evaluation using the Amazon MTurk crowd-sourcing service [29]. A similar training scheme is conducted in interaction with humans via the Amazon MTurk service and is shown to perform significantly better than a simulator trained policy. The final set of experiments discuss policy training in the full belief space where it is shown that the GP approach will scale to handle the full space obviating the need for hand-crafted summary space mapping. Related Gaussian process reinforcement learning approaches are then discussed in Section 5, while Section 6 presents conclusions and directions for future work.

3 Non-parametric policy modelling

Non-parametric policy modelling avoids the limitations on optimisation that occur when the solution is constrained by the chosen basis. A non-parametric representation does not necessarily mean that it is parameter-free, but rather that the solution is not restricted by the choice of parameters. Instead, the choice of parameters only influences the speed at which the optimal solution is found.

A Gaussian process is a non-parametric Bayesian model used for function approximation. It has been successfully applied to reinforcement learning for continuous-space MDPs, using both model-free approaches [30, 31] and model-based approaches [32, 33]. An advantage of Bayesian approaches to policy optimisation is that they offer a principled way of incorporating prior knowledge about the underlying task and this provides the potential to improve the learning rate. As already noted, it is important that the dependencies of dif-

ferent parts of the belief state space are taken into consideration during learning. Gaussian processes are helpful in that they are able to incorporate the knowledge of these dependencies elegantly through the choice of a *kernel function*, the purpose of which is to describe correlations in different parts of the space. In addition, Gaussian processes can provide the uncertainty of the approximation by estimating the variance of the posterior.

In the GP approach, the size of the state space only impacts the evaluation of the kernel. Hence, it is not sensitive to the size of the POMDP summary space and indeed it can be used to model correlations directly in the full belief space. In this paper, the properties of the GP approach are evaluated in summary space to provide consistency with other published results. Preliminary results using GP in the full belief space are also presented to demonstrate the potential for avoiding the need for hand-crafted summary space mapping functions.

This section provides an overview of the Gaussian process model of the Q-function based on the description given in [31]. Several policy formulations derived from the GP model are presented which are suitable for dialogue management. Issues of computational complexity and solutions which depend on sparse approximation are also discussed.

3.1 Gaussian process model for the Q-function

This section describes how the Q-function can be modelled as a Gaussian process. As outlined in the introduction, a discrete-space POMDP can be viewed as a continuous-space MDP. In dialogue management, however, this space is often reduced to a summary space that contains both continuous and discrete variables [17]. Therefore, in the most general case, the approximation framework needs to support modelling of the Q-function in a multidimensional space that consists of both continuous and discrete variables. A Gaussian process allows such modelling. For now, however, an MDP with a full continuous belief state space $\mathcal{B}$ is considered.

The discounted return R_t^π for time step t and a given policy π is the total accumulated reward acquired using policy π starting from the time step t:

$$R_t^\pi = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}, \tag{3}$$

where r_t is the immediate reward at time step t and $\gamma \in [0, 1]$, is the discount factor. If the immediate reward is a random process, the discounted return is also a random process.

The discounted return for policy π can be written recursively as:

$$R_t^\pi = r_{t+1} + \gamma R_{t+1}^\pi. \tag{4}$$

The Q-function for policy π is the expectation of the discounted return for that policy given belief state $\mathbf{b}$ and action a at time t, over all possible belief state sequences that can be generated with policy π (1):

$$Q^\pi(\mathbf{b}, a) = E_\pi \left(R_t | b(s_t) = \mathbf{b}, a_t = a \right). \tag{5}$$

If the transition probabilities do not change over time, $Q^\pi(\mathbf{b}, a)$ is not a random value, however, during the process of estimation it can be considered to be one. The estimate of the discounted return at time t can therefore be decomposed into a mean $Q^\pi(\mathbf{b}, a)$ and a residual $\Delta Q^\pi(\mathbf{b}, a)$:

$$R_t^\pi(b(s_t) = \mathbf{b}, a_t = a) = Q^\pi(\mathbf{b}, a) + \Delta Q^\pi(\mathbf{b}, a). \tag{6}$$

Substituting R_t^π and R_{t+1}^π from (6) into (4) yields:

$$\begin{aligned} r_{t+1}(b(s_t) = \mathbf{b}, a_t = a) \quad &= Q^\pi(\mathbf{b}, a) - \gamma Q^\pi(\mathbf{b}', a') \\ &+ \Delta Q^\pi(\mathbf{b}, a) - \gamma \Delta Q^\pi(\mathbf{b}', a'), \end{aligned} \tag{7}$$

where $b(s_{t+1}) = \mathbf{b}'$ is the next belief state and $a' = \pi(\mathbf{b}')$ is the next action, $a_{t+1} = a'$.

Let $\mathbf{B}_t = [(\mathbf{b}^0, a^0), \ldots, (\mathbf{b}^t, a^t)]^\mathsf{T}$ be a sequence of t belief states and action samples[1] generated with policy π. Then, (7) becomes:

$$\begin{aligned} r^1 \quad &= Q^\pi(\mathbf{b}^0, a^0) - \gamma Q^\pi(\mathbf{b}^1, a^1) \\ &+ \Delta Q^\pi(\mathbf{b}^0, a^0) - \gamma \Delta Q^\pi(\mathbf{b}^1, a^1) \\ r^2 \quad &= Q^\pi(\mathbf{b}^1, a^1) - \gamma Q^\pi(\mathbf{b}^2, a^2) \\ &+ \Delta Q^\pi(\mathbf{b}^1, a^1) - \gamma \Delta Q^\pi(\mathbf{b}^2, a^2) \\ &\vdots \\ r^t \quad &= Q^\pi(\mathbf{b}^{t-1}, a^{t-1}) - \gamma Q^\pi(\mathbf{b}^t, a^t) \\ &+ \Delta Q^\pi(\mathbf{b}^{t-1}, a^{t-1}) - \gamma \Delta Q^\pi(\mathbf{b}^t, a^t), \end{aligned} \tag{8}$$

where $r^1, \ldots, r^t$ are the acquired immediate rewards. This can be written in a more compact form as

$$\mathbf{r}_t = \mathbf{H}_t \mathbf{q}_t^\pi + \mathbf{H}_t \boldsymbol{\Delta} \mathbf{q}_t^\pi, \tag{9a}$$

where

$$\mathbf{r}_t = [r^1, \ldots, r^t]^\mathsf{T} \tag{9b}$$

$$\mathbf{q}_t^\pi = [Q^\pi(\mathbf{b}^0, a^0), \ldots, Q^\pi(\mathbf{b}^t, a^t)]^\mathsf{T}, \tag{9c}$$

$$\boldsymbol{\Delta} \mathbf{q}_t^\pi = [\Delta Q^\pi(\mathbf{b}^0, a^0), \ldots, \Delta Q^\pi(\mathbf{b}^t, a^t)]^\mathsf{T}, \tag{9d}$$

and

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & -\gamma \end{bmatrix}. \tag{9e}$$

By modelling $Q(\mathbf{b}, a)$ as a Gaussian process, $Q(\mathbf{b}, a) \sim \mathcal{GP}\left(0, k((\mathbf{b}, a), (\mathbf{b}, a))\right)$, where the kernel $k(\cdot, \cdot)$ is factored into separate kernels over the belief state and action spaces $k_\mathcal{B}(\mathbf{b}, \mathbf{b}')k_\mathcal{A}(a, a')$ and $\Delta Q^\pi(\mathbf{b}, a) \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise, a similar procedure to Gaussian process regression (see Appendix 1) can be applied to find the posterior of $Q^\pi(\mathbf{b}, a)$, given a set of belief state-action pairs $\mathbf{B}_t$ and the observed rewards $\mathbf{r}_t$ in these belief states:

$$\begin{aligned} &Q^\pi(\mathbf{b}, a) | \mathbf{r}_t, \mathbf{B}_t \sim \mathcal{N}(\overline{Q}(\mathbf{b}, a), \mathrm{cov}((\mathbf{b}, a), (\mathbf{b}, a))), \\ &\overline{Q}(\mathbf{b}, a) = \mathbf{k}_t(\mathbf{b}, a)^\mathsf{T} \mathbf{H}_t^\mathsf{T} (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\mathsf{T} + \sigma^2 \mathbf{H}_t \mathbf{H}_t^\mathsf{T})^{-1} \mathbf{r}_t, \\ &\mathrm{cov}((\mathbf{b}, a), (\mathbf{b}, a)) = k((\mathbf{b}, a), (\mathbf{b}, a)) \\ &\quad - \mathbf{k}_t(\mathbf{b}, a)^\mathsf{T} \mathbf{H}_t^\mathsf{T} (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\mathsf{T} + \sigma^2 \mathbf{H}_t \mathbf{H}_t^\mathsf{T})^{-1} \mathbf{H}_t \mathbf{k}_t(\mathbf{b}, a) \end{aligned} \tag{10}$$

[1] Random variables are denoted with a subscript, *e.g.*, a_t is a random variable at some time step t. Samples are denoted with a superscript *e.g.*, a^t is the action that was taken at time step t.

where $\mathbf{k}_t(\mathbf{b}, a) = [k((\mathbf{b}^0, a^0), (\mathbf{b}, a)), \ldots, k((\mathbf{b}^t, a^t), (\mathbf{b}, a))]^\mathsf{T}$ and
$\mathbf{K}_t = [\mathbf{k}_t((\mathbf{b}^0, a^0)), \ldots, \mathbf{k}_t((\mathbf{b}^t, a^t))]$.

This Gaussian process model of the Q-function exploits the relationship between *distributions* of the Q-function at differing time steps. This is in contrast to standard reinforcement learning algorithms which typically exploit the relationship between *values* of the Q-function at differing time steps. Initially, the estimated Q-function distribution is just a zero-mean Gaussian process with the kernel function $k((\mathbf{b}, a), (\mathbf{b}, a))$, since no data has been observed. By time step t, the estimate is the posterior distribution given the set of observed rewards $\mathbf{r}_t$ and associate belief state-action pairs $\mathbf{B}_t$.

It can be shown [34] that the marginal likelihood of the observed rewards is modelled by

$$\mathbf{r}_t | \mathbf{B}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{H}_t(\mathbf{K}_t + \sigma^2 \mathbf{I})\mathbf{H}_t^\mathsf{T}). \tag{11}$$

If the kernel function is parametrised, this relationship can be used to optimise the kernel parameters as well as the noise parameter σ [35].

In the next section the derivation of a policy model from the Gaussian process model of the Q-function is discussed.

3.2 Gaussian process-based policy model

The description so far has presented a Gaussian process model for the Q-function associated with a policy π given belief state-action pairs $\mathbf{B}_t$ and rewards $\mathbf{r}_t$ (10). What remains is to define a policy based on this model.

The most obvious way of deriving such a policy is simply to select the action associated with the highest mean of the Q-function given any belief state $\mathbf{b}$:

$$\pi(\mathbf{b}) = \arg\max_a \overline{Q}(\mathbf{b}, a). \tag{12}$$

However, since the objective is to learn the Q-function associated with the optimal policy by interacting directly with users, the policy must exhibit some form of stochastic behaviour in order to explore alternatives during the process of learning. One way of defining a policy that can be optimised on-line is to use an ϵ-greedy approach. This requires setting an additional parameter ϵ which balances how often an action is taken according to the current best estimate of the Q-function mean (the *exploitation* stage) compared to how often an action is taken randomly (the *exploration* stage). This ϵ-greedy policy is defined as:

$$\pi(\mathbf{b}) = \begin{cases} \arg\max_a \overline{Q}(\mathbf{b}, a) & \text{with prob } 1 - \epsilon, \\ \text{random } a & \text{with prob } \epsilon. \end{cases} \tag{13}$$

However, such random exploration can be inefficient since not all parts of belief-action space are equally informative. Using active learning to provide more efficient exploration should yield faster learning [36]. Instead of selecting actions randomly, active learning selects actions according to some *utility function* which normally includes some measure of information gain through which various heuristics can be incorporated [37]. A particularly appealing aspect of Gaussian process reinforcement learning is that it provides a measure of uncertainty at each point in belief-action space. This uncertainty can then be used directly in the active learning utility function [33]. This enables the model to explore the parts of the space it is less certain about. Therefore, during exploration, actions are chosen according to the variance of the GP estimate for the Q-function and during exploitation, actions are chosen according to the mean:

$$\pi(\mathbf{b}) = \begin{cases} \arg\max_a \overline{Q}(\mathbf{b}, a) & \text{with prob } 1 - \epsilon, \\ \arg\max_a \operatorname{cov}((\mathbf{b}, a), (\mathbf{b}, a)) & \text{with prob } \epsilon. \end{cases} \tag{14}$$

Both of the policies defined in (13) and (14) require the proportions of exploration and exploitation to be balanced manually by tuning the parameter ϵ. Since the Gaussian process for the Q-function defines a Gaussian distribution for every belief state-action pair (10), when a new belief point $\mathbf{b}$ is encountered, for each action $a \in \mathcal{A}$, there is a Gaussian distribution $\hat{Q}(\mathbf{b}, a) \sim \mathcal{N}(\overline{Q}(\mathbf{b}, a), \mathrm{cov}((\mathbf{b}, a), (\mathbf{b}, a)))$. Sampling from these Gaussian distributions gives a set of Q-values from which the action with the highest sampled Q-value can be selected:

$$\pi(\mathbf{b}) = \arg\max_a \left\{ \hat{Q}(\mathbf{b}, a) : a \in \mathcal{A} \right\}. \tag{15}$$

Thus, in contrast to the ϵ-greedy policies defined by (13) and (14), this approach maps the GP approximation of the Q-function into a stochastic policy model. Note however that all three policies are *greedy* in the sense that as either ϵ or the variance tends to zero, they tend to select the locally optimal action. The performance of these three policy models is discussed further below in section 4.2.

3.3 Gaussian process-based policy optimisation

If the Q-function of a greedy policy is estimated on-line using temporal differences, then it will in the limit tend to the Q-function of the optimal policy. A commonly used embodiment of this idea for MDPs with observable states is the Sarsa algorithm which iteratively updates the Q-function on-line using the rule:

$$\begin{aligned} Q(\mathbf{b}, a) \leftarrow\ & Q(\mathbf{b}, a) \\ & + \alpha[r_{t+1}(b_t = \mathbf{b}, a_t = a) + \gamma Q(\mathbf{b}', a') - Q(\mathbf{b}, a)]. \end{aligned} \tag{16}$$

If the simple Q-function correction provided by the right hand side of (16) is replaced by the update of the GP-based Q-function posterior given by (10), the GP-Sarsa algorithm is obtained. This is shown in Fig. 1 amended for episodic reinforcement learning and hence applicable to spoken dialogue optimisation.

In practice, however, the exact computation of the posterior using (10) is intractable since it requires the inversion of a matrix of dimensionality t. A solution to this problem is discussed in the next section.

3.4 Computational complexity and sparse approximation

Due to the matrix inversion in (10), the computational complexity of calculating the Q-function posterior is $O(t^3)$, where t is the number of data points. In the case of a dialogue system, the number of points used for estimation will be equal to the total number of turns, summed over all dialogues. This poses a serious potential computational problem, since the total number of turns needed to accurately estimate the optimal policy will be large.

One solution to this problem is to restrict the set of data points used for the Q-function approximation. This has the obvious drawback that useful information will inevitably be discarded and this is exacerbated when the space is continuous (as here) since the exact same state is never visited twice. To mitigate the effects of restricting the number of data points, it is therefore important to take into account the contribution each part of the belief state space brings to the final estimation. Sparse approximation methods for Gaussian processes seek to achieve this such that all data points are taken into account whilst reducing the computational complexity.

The cubic increase in complexity with the number of visited belief states is the Achilles' heel of Gaussian process regression. A significant research effort has been invested into solving this problem, and a number of methods have been developed [38]. However, these rely on the pre-selection of a fixed set of m support points. This reduces the complexity

Figure 1: Episodic GP-Sarsa (without computational constraints)

for each episode **do**
  Initialise $\mathbf{b}$
  if first episode **then**
    Choose a arbitrary
    $\mathbf{B}_0 \leftarrow (\mathbf{b}, a)$
    $\mathbf{K}_0 \leftarrow k((\mathbf{b}, a), (\mathbf{b}, a))$
    $\mathbf{H}_0 \leftarrow \begin{bmatrix} 1 & -\gamma \end{bmatrix}$
  else
    if initial step **then**
      Choose $a \leftarrow \pi(\mathbf{b})$ (13), (14) or (15)
    end if
  end if
  for each step in the episode **do**
    Take a, observe r', update $\mathbf{b}'$
    if non-terminal step **then**
      Choose new action $a' \leftarrow \pi(\mathbf{b}')$ (13), (14) or (15)
      $\mathbf{B}_{t+1} \leftarrow \begin{bmatrix} \mathbf{B}_t & (\mathbf{b}', a') \end{bmatrix}$
      $\mathbf{H}_{t+1} \leftarrow \begin{bmatrix} \mathbf{H}_t & \mathbf{0} \\ \mathbf{u}^\mathsf{T} & -\gamma \end{bmatrix}$, where $\mathbf{u} = \begin{bmatrix} \mathbf{0} & 1 \end{bmatrix}^\mathsf{T}$
      $\mathbf{K}_{t+1} \leftarrow \begin{bmatrix} \mathbf{K}_t & \mathbf{k}_t(\mathbf{b}', a') \\ \mathbf{k}_t(\mathbf{b}', a') & k(\mathbf{b}', a', \mathbf{b}', a') \end{bmatrix}$
    else
      $\mathbf{B}_{t+1} \leftarrow \mathbf{B}_t$, $\mathbf{K}_{t+1} \leftarrow \mathbf{K}_t$
      $\mathbf{H}_{t+1} \leftarrow \begin{bmatrix} \mathbf{H}_t \\ \mathbf{u}^\mathsf{T} \end{bmatrix}$, where $\mathbf{u} = \begin{bmatrix} \mathbf{0} & 1 \end{bmatrix}^\mathsf{T}$
    end if
    $\mathbf{r}_{t+1} \leftarrow \begin{bmatrix} \mathbf{r}_t & r' \end{bmatrix}$
    Update Q-function posterior $Q^\pi | \mathbf{r}_{t+1}, \mathbf{B}_{t+1}$ (10)
    if non-terminal step **then**
      $\mathbf{b} \leftarrow \mathbf{b}'$, $a \leftarrow a'$
    end if
  end for
end for

of calculating the posterior from $O(t^3)$ to $O(tm^2)$ and if the number of support points is significantly smaller than the number of data points, this approach is very effective for reducing the computational cost. This class of methods is not useful for learning on-line, however, since the support points cannot be determined *a priori*. An alternative algorithm which approximates the Gaussian process without first obtaining a set of support points is the kernel span sparsification method described in [39]. In this case, a representative set of data points is acquired as the belief-action space is traversed during interaction with the user.

A kernel function can be thought of as the dot product of a (potentially infinite) set of feature functions $k((\mathbf{b}, a), (\mathbf{b}, a)) = \langle \boldsymbol{\phi}(\mathbf{b}, a), \boldsymbol{\phi}(\mathbf{b}, a) \rangle$ where $\boldsymbol{\phi}(\mathbf{b}, a)$ is the vector of feature functions $[\phi_1(\mathbf{b}, a), \phi_2(\mathbf{b}, a), \ldots]^\mathsf{T}$. Any linear combination of feature vectors $\{\boldsymbol{\phi}(\mathbf{b}^0, a^0), \ldots, \boldsymbol{\phi}(\mathbf{b}^t, a^t)\}$ for a given set of points $\{(\mathbf{b}^0, a^0), \ldots, (\mathbf{b}^t, a^t)\}$ is called the *kernel span*. The aim then is to find the subset of points that approximates this kernel span. These points are called the *representative points* and the set of representative points is called the

dictionary $\mathcal{D} = \{(\tilde{\mathbf{b}}^0, \tilde{a}^0), \ldots, (\tilde{\mathbf{b}}^m, \tilde{a}^m)\}$.

A sparsification parameter ν places a threshold on the squared distance between the feature function span at the representative points, and the true feature function value at each visited point. Every time the threshold is exceeded, a new point is added to the dictionary. If $(\mathbf{b}^t, a^t)$ is the current data point then:

$$\min_{\mathbf{g}_t} \| \sum_{j=0}^{m} g_{tj} \boldsymbol{\phi}(\tilde{\mathbf{b}}^j, \tilde{a}^j) - \boldsymbol{\phi}(\mathbf{b}^t, a^t) \|^2 \leq \nu, \tag{17}$$

where $\mathbf{g}_t = [g_{t1}, \ldots, g_{tj}]$ is a vector of coefficients and m is the size of the current dictionary, $\mathcal{D} = \{(\tilde{\mathbf{b}}^0, \tilde{a}^0), \ldots, (\tilde{\mathbf{b}}^m, \tilde{a}^m)\}$. It can be shown [39] that this is equivalent to:

$$\min_{\mathbf{g}_t} \left(k((\mathbf{b}^t, a^t), (\mathbf{b}^t, a^t)) - \tilde{\mathbf{k}}_{t-1}(\mathbf{b}^t, a^t)^\mathsf{T} \mathbf{g}_t \right) \leq \nu, \tag{18}$$

where $\tilde{\mathbf{k}}_{t-1}(\mathbf{b}^t, a^t) = [k((\mathbf{b}^t, a^t), (\tilde{\mathbf{b}}^0, \tilde{a}^0)), \ldots, k((\mathbf{b}^t, a^t), (\tilde{\mathbf{b}}^m, \tilde{a}^m))]^\mathsf{T}$. It can also be shown that the expression on the left side of (18) is minimised when $\mathbf{g}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{b}^t, a^t)$, where $\tilde{\mathbf{K}}_{t-1}$ is the Gram matrix of the current set of representative points. If the threshold ν is exceeded then $(\mathbf{b}^t, a^t)$ is added to the dictionary, otherwise the dictionary stays the same. This constitutes the sparsification criterion and it allows the posterior to be approximated with bounded complexity as follows.

Since the kernel function is the dot product of the feature functions, the Gram matrix is $\mathbf{K}_t = \boldsymbol{\Phi}_t^\mathsf{T} \boldsymbol{\Phi}_t$ where $\boldsymbol{\Phi}_t = [\boldsymbol{\phi}(\mathbf{b}^0, a^0), \ldots, \boldsymbol{\phi}(\mathbf{b}^t, a^t)]$. The feature function values for each point are then approximated as the linear combination of the representative points $\boldsymbol{\phi}(\mathbf{b}^i, a^i) \approx \sum_{j=1}^{m} g_{ij} \boldsymbol{\phi}(\tilde{\mathbf{b}}^j, \tilde{a}^j)$, for all $i \in 0, \ldots, t$. Also, the Gram matrix is approximated as

$$\mathbf{K}_t = \boldsymbol{\Phi}_t^\mathsf{T} \boldsymbol{\Phi}_t \approx \mathbf{G}_t \tilde{\mathbf{K}}_t \mathbf{G}_t^\mathsf{T}, \tag{19}$$

where $\mathbf{G}_t = [\mathbf{g}_1, \ldots, \mathbf{g}_t]$. In a similar manner, $\mathbf{k}_t(\mathbf{b}, a) \approx \mathbf{G}_t \tilde{\mathbf{k}}_t(\mathbf{b}, a)$.

This allows the posterior (10) to be approximated as:

$$\begin{aligned}
&Q(\mathbf{b}, a) | \mathbf{B}_t, \mathbf{r}_t \sim \mathcal{N}\left(\widetilde{\overline{Q}}(\mathbf{b}, a), \widetilde{\mathrm{cov}}((\mathbf{b}, a), (\mathbf{b}, a)) \right), \\
&\widetilde{\overline{Q}}(\mathbf{b}, a) = \tilde{\mathbf{k}}_t(\mathbf{b}, a)^\mathsf{T} (\tilde{\mathbf{H}}_t^\mathsf{T} (\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\mathsf{T} + \sigma^2 \tilde{\mathbf{H}}_t \tilde{\mathbf{H}}_t^\mathsf{T})^{-1} \mathbf{r}_t), \\
&\widetilde{\mathrm{cov}}((\mathbf{b}, a), (\mathbf{b}, a)) = k((\mathbf{b}, a), (\mathbf{b}, a)) \\
&\quad - \tilde{\mathbf{k}}_t(\mathbf{b}, a)^\mathsf{T} (\tilde{\mathbf{H}}_t^\mathsf{T} (\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\mathsf{T} + \sigma^2 \tilde{\mathbf{H}}_t \tilde{\mathbf{H}}_t^\mathsf{T})^{-1} \tilde{\mathbf{H}}_t) \tilde{\mathbf{k}}_t(\mathbf{b}, a),
\end{aligned} \tag{20}$$

where $\tilde{\mathbf{H}}_t = \mathbf{H}_t \mathbf{G}_t$. It can be shown that this reduces the complexity to $O(tm^2)$, where m is the number of representative points.

This sparsification method can be effectively utilised in an on-line algorithm where the data is obtained sequentially. The process starts with just one data point which constitutes the initial dictionary. Every time a new data point is encountered, it is checked if the kernel at that point can be approximated as the kernel span of the current dictionary points up to the precision defined by the sparsification threshold ν. If the threshold is exceeded, the new data point is added to the dictionary.

However, this sparsification method turns a non-parametric method into a parametric method. More precisely, the kernel is now approximated using only a limited number of points, which is equivalent to defining a functional basis for the kernel function. Defining the kernel to be a finite linear combination of that basis can be shown to be equivalent to parameterising the Q-function [31]. This may limit the accuracy of the solution since the method finds the optimal solution only within the span of the kernel basis functions.

However, the fact that the basis is chosen dynamically, without an initial set of basis functions, still allows for more appropriate functions than when using a fixed basis defined by a designer.

A further advantage of this sparsification approach is that it enables non-positive definite kernel functions to be used in the approximation. This is due to the fact that the sparsification method essentially changes the kernel function in a manner which ensures that the approximated Gram matrix is positive definite and this is sufficient to guarantee that the model remains well-defined [40]. Furthermore, using a sparse approximation sometimes yields better results than the full Gaussian process model since the approximated kernel function might be a better fit than the original kernel function [41].

This sparsification method allows observations to be processed sequentially, in direct interaction with the user, whether real or simulated and it can be incorporated directly into the GP-Sarsa algorithm out-lined in Fig. 1. The full GP-Sarsa algorithm with kernel sparsification is given in Appendix 2.

4 Experiments

The previous sections have introduced the theory of Gaussian process-based POMDP policy optimisation. Here we examine their application in practice and investigate three important research issues. Firstly, section 4.2 addresses the detailed design trade-offs involved in practical implementation of a GP-based dialogue policy including the kernel function and policy model. Secondly, section 4.3 investigates the speed of the GP policy optimisation process and whether in practice it can be performed in direct interaction with real users. Finally, section 4.4 addresses the extent to which design time effort can be reduced by applying a GP-policy directly to the full belief space and hence avoiding the need for *ad hoc* summary space mapping.

4.1 Experimental set-up

A spoken dialogue system providing restaurant information for Cambridge UK was used as a test-bed for all of the reported experiments. This system uses the BUDS dialogue manager in which beliefs are modelled by a Bayesian network. Training and testing was provided by an agenda-based simulated user and by real users recruited using the Amazon MTurk service.

Bayesian Update of Dialogue State dialogue manager

The Bayesian Update of Dialogue State (BUDS) dialogue manager is a POMDP-based dialogue manager [24] whose belief state consists of the marginal posterior probability distribution over hidden nodes in a Bayesian network. Each concept in the dialogue, eg. *area*, *food-type*, *address* is represented by a pair of hidden nodes recording the history and the goal. The history nodes define possible states for a particular concept, eg. *system-informed*, *user-informed* and the goal nodes define possible values for a particular concept, eg. *Chinese*, *Indian*.

In order to train an optimal policy the following approach is normally taken. The belief space $\mathcal{B}$ is mapped into a summary space $\mathcal{C}$. Likewise, the action space is mapped into a smaller scale summary action space $\mathcal{A}$. This summary space $\mathcal{C} \times \mathcal{A}$ is then mapped into a feature space $\mathcal{F}$. The feature space is then used to produce a parametric representation of the policy. A weighted set of these basis functions can then be cast into a probability via a

soft-max function as in

$$\pi(a|\mathbf{c};\theta) = \frac{e^{\theta \cdot f_a(\mathbf{c})}}{\sum_a e^{\theta \cdot f_a(\mathbf{c})}}. \tag{21}$$

where $f_a(\mathbf{c})$ is the vector of features derived from summary state $\mathbf{c}$ for action a. The policy is then optimised using the natural actor critic (NAC) algorithm which is a gradient-based method [42]. In operation, system responses are generated by sampling (21) and then heuristically mapping the summary action back into a full system action using information from the full belief space.

Mappings $\mathcal{B} \rightarrow \mathcal{C}$ and $\mathcal{C} \times \mathcal{A} \rightarrow \mathcal{F}$ require a significant amount of hand-crafting. Also, due to the parametric policy representation, the solution is only optimal within the given basis. Finally, gradient-based optimisation methods are inherently slow which prevents direct on-line optimisation with real users. Here we show that the GP-Sarsa algorithm can overcome these limitations.

The Cambridge restaurant domain

The Cambridge restaurant domain consists of a selection of approximately 150 restaurants that have been automatically extracted from various web-based sources. Each restaurant has 8 attributes (slots) and this results in a belief space consisting of 25 concepts where each concept takes from 3 to 150 values and each value has a probability in $[0, 1]$. The summary space is formed from discrete, as well as continuous, features. Continuous features represent the entropy of the distribution for each hidden node in the Bayesian network. In addition, there are a few discrete features that correspond to history nodes, e.g. if the highest probability method that the user used to inform the system was *by name of the venue* or if the highest probability discourse act used was *repeat*. Also, there is a count of the number of top probability goal values that are greater than 0.8. Finally, there is a list which defines the order in which the goal slots can be accepted. In total, there are 129 features. The summary action space consists of 16 summary actions.

The agenda-based simulated user

In training and testing, an agenda-based user simulator was used [43, 44]. The user state is factored into an *agenda* and a *goal*. The goal ensures that the user simulator exhibits consistent, goal-directed behaviour. The role of the agenda is to elicit the dialogue acts that are needed for the user simulator to fulfil the goal. Both the goal and the agenda are dynamically updated throughout the dialogue. These updates are controlled by *decision points*. Decision points can be deterministic or stochastic. The stochastic decision points are controlled with parametrised probability distributions. In this way, they enable a wide spread of realistic dialogues to be generated.

In addition, an error model was used to add confusions to the simulated user input such that it resembles those found in real data [45]. The output of the error model is an N-best list of possible user responses. The length of this list was set to 10 and the confusion rate was set to 15%, which means that 15% of time the true hypothesis is not in the N-best list. Intermediate experiments showed that these settings match the confusions typically found in real data. Unless otherwise stated, the same confusion rate was used both during training and testing.

The reward function was set to give a reward of 20 for successful dialogues, zero otherwise. In addition, one is deducted for each dialogue turn. The discount factor γ is set to 1 and the dialogue length is limited to 30 turns.

The evaluation schedule

When evaluating a policy optimisation technique one is typically interested in the performance of the resulting policy. However, it is also important to know how many dialogue sessions are needed to reach that performance. In other words, if the training was stopped earlier, what would the policy performance be? Traditionally, reinforcement learning algorithms suffer from the fact that not only do they need a large number of dialogues to train a dialogue policy, but they also perform poorly at the beginning of the training. This makes them unsuitable for training in direct interaction with real users. Finally, policy optimisation is a random process itself. A lucky choice of actions at the beginning may lead to very good performance, whereas an unlucky choice of actions can slow down the process of learning. Therefore, it is important to examine how much variability in performance one can expect during training.

For this reason, in all the user simulator-based experiments reported here, unless stated otherwise, 10 training sessions were seeded with different random seeds and after every 1000 dialogues the performance of the partially trained policies were evaluated. Each evaluation was performed with 1000 dialogues ensuring that the simulated user had a different random seed to the seed used in training mode. The results were then averaged and presented with error bars of one standard error.

The MTurk Amazon Service

In order to evaluate different policies with real people we recruited subjects using the Amazon MTurk crowd-sourcing service in a set-up similar to [46]. The BUDS dialogue manager was incorporated in a live telephone-based spoken dialogue system in which human users were assigned specific tasks which involved finding restaurants that have particular features. To elicit more complex dialogues, users were sometimes asked to find more than one restaurant, or a restaurant that does not exist. In the latter case, they were asked to change one of their constraints, for example find a Chinese restaurant instead of a Vietnamese one. After each dialogue, users filled in a feedback form indicating whether they thought the dialogue was successful. Based on this rating, the subjective success was calculated as well as the average reward.

4.2 Design of Gaussian process-based policy optimisation

In order to use the GP-Sarsa algorithm, the kernel function, the noise parameter σ and the sparsification parameter ν need to be defined. Whilst the choice of the kernel function and the choice of σ do not greatly affect the resulting policy, they do have considerable influence on the speed of learning. In addition, the parameter ν controls the approximation of the kernel function, essentially turning a non-parametric optimisation into a parametric one, and if set inappropriately it can lead to the resulting policy being suboptimal. Below we examine the different design choices.

Kernel function

The kernel function defines prior correlations in different parts of the summary space and if correctly defined can significantly speed up the process of learning. It ideally should be defined to suit the particular domain. However, there are a number of standard kernel functions that can be used.

As explained in Section 4.1, the summary space $\mathcal{C}$ consists of both continuous and discrete variables. We therefore define the kernel function on $\mathcal{C}$ as the sum of kernels on

continuous space and discrete space:

$$k_{\mathcal{C}}(\mathbf{c}, \mathbf{c}') = k_{\text{cont}(\mathcal{C})}(\mathbf{c}, \mathbf{c}') + k_{\text{disc}(\mathcal{C})}(\mathbf{c}, \mathbf{c}') \tag{22}$$

We investigated two standard kernel functions on the continuous parts of the summary space.

The polynomial kernel function is defined as:

$$k(\mathbf{c}, \mathbf{c}'; \sigma_0, p) = (\langle \mathbf{c}, \mathbf{c}' \rangle + \sigma_0^2)^p, \tag{23}$$

where $\langle \cdot, \cdot \rangle$ is the dot-product, with hyper-parameters $\sigma_0 >= 0$ and $p > 0$. In the case where $p = 1$ this is the linear kernel. Higher order polynomial kernels can lead to better results for small input spaces [34]. However, they are less suitable for large input spaces such as the one used here since the prior variance $k(\mathbf{c}, \mathbf{c})$ grows rapidly with $|\mathbf{c}| > 1$ [47].

The Gaussian kernel function[2] is defined as:

$$k(\mathbf{c}, \mathbf{c}'; p, \sigma_k) = p^2 \exp\left(- \frac{\|\mathbf{c} - \mathbf{c}'\|^2}{2\sigma_k^2} \right), \tag{24}$$

where σ_k determines how close the points have to be for the values of the function to be correlated, and p defines the prior variance at each data point since $k(\mathbf{c}, \mathbf{c}) = p^2$. The main advantage of the Gaussian kernel over the polynomial kernel is that it is a dot product of an infinite vector of feature functions [47], which gives it the potential to model covariances better. On the other hand, the Gaussian kernel is a stationary kernel meaning that that the covariances only depend on the distance between two summary states. This is in contrast to the polynomial kernel which is non-stationary since the covariance depends on the part of the space where the two summary states are located.

The GP-Sarsa algorithm requires that the kernel function is defined on the summary actions space too. Since this is a small discrete space, the δ-kernel was used:

$$k(a, a') = 1 - \delta_a(a'). \tag{25}$$

The same function is also used on discrete parts of the summary space.

Fig. 2 compares a linear kernel ($\sigma_0 = 1$ and $p = 1$) with a Gaussian kernel ($\sigma_k = 5$ and $p = 4$). The results suggest that the Gaussian kernel is better suited for this problem as it results in training that is less variable.

Residual noise

The second element of policy design is the choice of parameter σ, the noise of the residual ΔQ (see Section 3.1). As already noted, this controls how much change in the estimate of the Q-function is expected to take place during the process of learning. It is directly related to the randomness of the return (3) and therefore depends on the reward function. The rule of thumb is that σ should be the square root of the half length of the interval over possible values of the reward function. The intuition behind this is that the return in principle can take any value between the highest and the lowest possible reward. Since the process of learning is non-stationary, the return at some point in the state space for the same action can vary significantly during the optimisation process and the model should therefore allow for this variation. Therefore, defining ΔQ to have a Gaussian distribution with a variance which can cover the whole reward interval seems reasonable. The reward function used here gives 20 for a successful dialogue less the number of turns, where the user can take up to

[2] Also called the squared exponential kernel function in the literature.

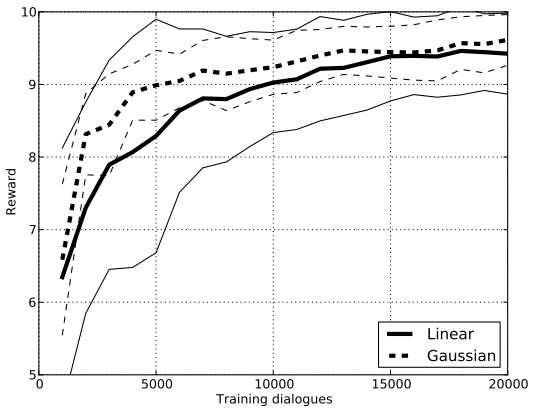

Figure 2: Influence of the kernel choice on the performance. The performance is measured as the mean average reward as a function of the number of training dialogues. The thinner lines denote one standard error.

30 turns. Therefore, the interval is $[-30, 20]$ so σ is fixed at 5. To experimentally support this claim, values of $\sigma = 1$, $\sigma = 5$ and $\sigma = 10$ were tested and the results are shown in Fig 3 where it can be seen that $\sigma = 5$ does indeed give the best performance.

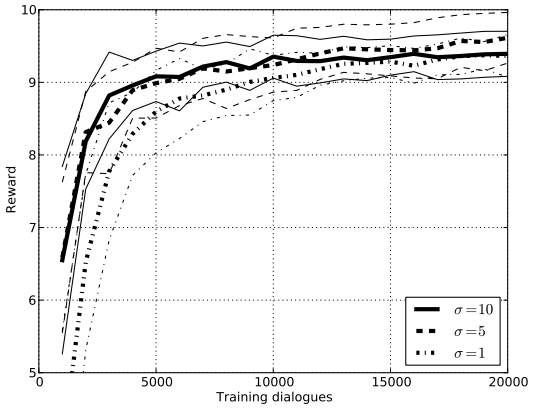

Figure 3: Influence of the choice of the noise residual parameter σ on the performance. The thinner lines denote one standard error.

Sparsification threshold

The sparsification threshold ν represents a trade-off between the computational complexity and the precision to which the kernel function is computed. The computational complexity is m^2 at every dialogue turn, where m is the total number of dictionary points. Fig. 4 compares performance for $\nu = 0.1$, $\nu = 1$ and $\nu = 10$. As can be seen, the smaller ν is the better performance is, which intuitively makes sense. It is interesting to note however that the difference in performance between $\nu = 0.1$ and $\nu = 1$ is almost negligible in comparison

to the difference between $\nu = 1$ and $\nu = 10$. Fig. 5 gives the number of dictionary points for different thresholds. The average execution times for each of these settings are given in Fig. 6[3]. Comparing the two, it can be seen that for a very small improvement in performance the computational complexity dramatically increases, because of the increase in the number of dictionary points.

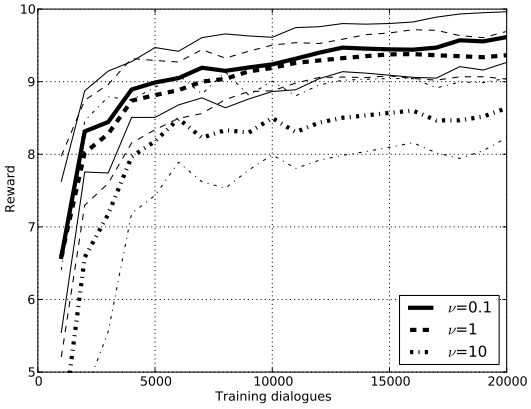

Figure 4: Influence of the sparsification threshold ν on the performance. The thinner lines denote one standard error.

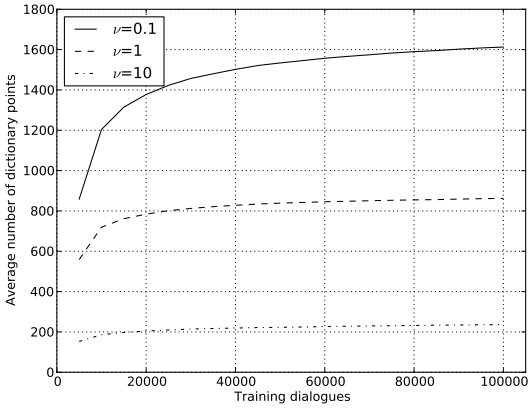

Figure 5: Average number of dictionary points for different settings of ν as a function of the number of training dialogues.

Policy model

As already noted in Section 3.2, the policy model can be defined ϵ-greedily (13), using active learning (14), or stochastically (15). It has already been shown that active learning has the

[3]The runtime results are obtained on a 64 core AMD Opteron(TM) Processor 6276 running at 2.29GHz with 500Gbytes RAM.

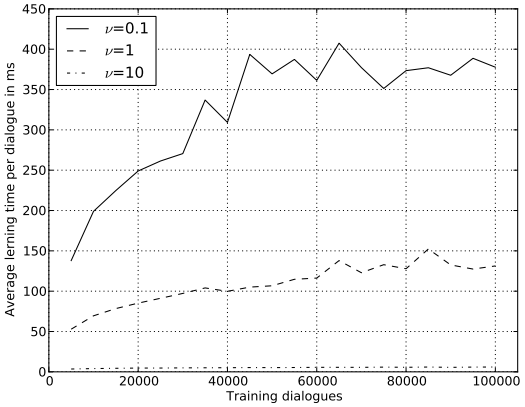

Figure 6: Average compute time for different settings of ν during learning as a function of the number of training dialogues.

potential to lead to faster learning [35] as well as other learning strategies that exploit the variance [48, 49]. Here we focus on the stochastic policy since it avoids the need to explicitly control the exploration rate during training, and it provides a uniform mechanism for both optimisation and normal run-time operation.

Gaussian process estimation has an inherent problem in that the estimate of the variance depends only on the number of points from the input space rather than their values in the output space. This is exacerbated in the set-up used here since the form of Gaussian process approximation used can lead to overly confident estimates [38] whilst at the same time, as input spaces become larger, the same points need to be visited more and more times in order to achieve optimal performance. Hence, when defining a stochastic policy for a large input space, such as the one used here, it is helpful to scale the variances during training to ensure an adequate degree of exploration: $Q(\mathbf{b}, a) \sim \mathcal{N}(\overline{Q}(\mathbf{b}, a), \eta^2 \text{cov}((\mathbf{b}, a), (\mathbf{b}, a))))$ where η is the scaling factor.

Figs. 7 and 9 show performance during training and testing, respectively, for a stochastic policy with variances scaled by $\eta = 1$, $\eta = 2$ and $\eta = 3$.[4] For comparison, the performance of an ϵ-greedy policy is also shown as a baseline. In addition, the standard error during training is shown in Fig. 8.

As can be seen, varying the scale factor η has a significant influence on performance. Setting η to a low value results in the system learning quickly, but also converging to a local optimum and, as shown in Fig. 8, subsequent performance is very variable. Whilst the best stochastic policy model shows only a small improvement in comparison to the ϵ-greedy learning during testing (Fig. 9), it is important to note that the performance during training is significantly higher than ϵ-greedy learning (Fig. 7). This is particularly important when training in interaction with real users since poor performance during the early stages of training may impact on the user's willingness to continue to interact with the system.

4.3 Fast policy optimisation

One of the claimed advantages of the GP-Sarsa algorithm over other reinforcement learning algorithms is its ability to learn from a small amount of data, exploiting the correlations

[4] Higher values of η did not produce any further improvements in performance.

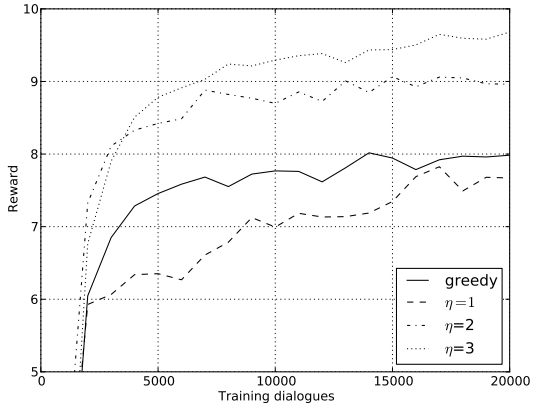

Figure 7: Influence of variance scaling and the choice of the policy model on the performance during training.

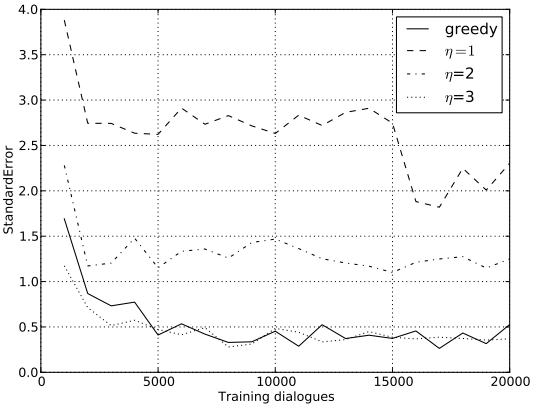

Figure 8: Standard error of the average reward during training of different policy models.

defined by the kernel function. In this section, this claim is evaluated by comparing the GP-Sarsa learning rate with the NAC gradient-based algorithm using a user simulator, and through on-line learning with real users.

Comparison with standard methods

It has been previously shown that GP-Sarsa outperforms standard non-parametric policy approaches [35]. Here it is compared to natural actor critic (NAC) learning, a parametric gradient-based reinforcement learning algorithm.

NAC policy optimisation for the BUDS dialogue manager relies on the parametric representation of the policy π using a set of feature functions $\mathcal{F}$, as explained in Section 4.1. The GP-Sarsa algorithm directly optimises the policy in the summary state $\mathcal{C}$. Both GP-Sarsa and NAC used training procedures with exactly the same number of dialogues in interaction with the simulated user.

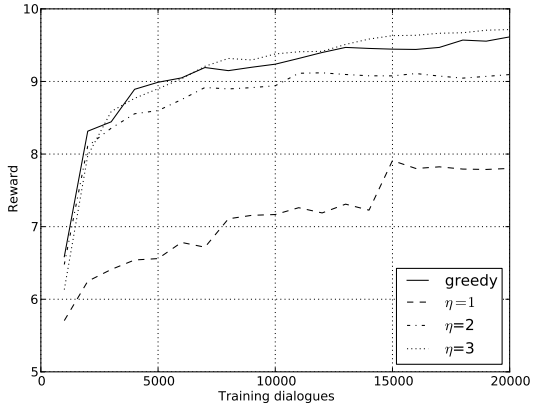

Figure 9: Influence of variance scaling and the choice of the policy model on the performance during testing.

To measure the rate at which each algorithm learns, the partially optimised policies were evaluated on 1000 random dialogues after being trained on 5000 dialogues. This process was repeated until the policies had been trained with 100,000 dialogues in total. The confusion rate was set to 15% throughout. The simulated user had a different random seed in testing mode to the one set in training mode. The average reward, the success rate and the average number of turns are shown in Figs. 10, 11 and 12 respectively. The error bars represent one standard error.

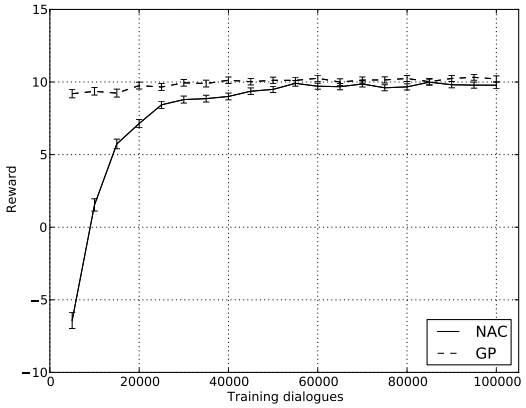

Figure 10: Comparison of the performance of GP-Sarsa vs the natural actor critic algorithm, where the performance is measured as the average reward as a function of the number of training dialogues.

The results show that the GP-based policy reaches a performance comparable to the fully trained NAC policy in only 10,000 dialogues. This is consistent with related research which has shown that Gaussian processes can be deployed to guide gradient descent and improve its efficiency [50]. Also, the fully trained GP policy outperforms the NAC policy

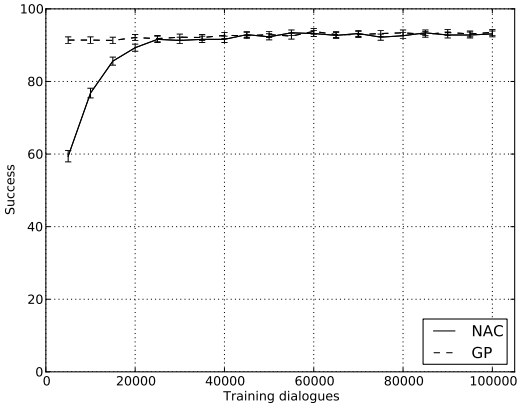

Figure 11: Comparison of the performance of GP-Sarsa vs the natural actor critic algorithm, where the performance is measured as the average success as a function of the number of training dialogues.

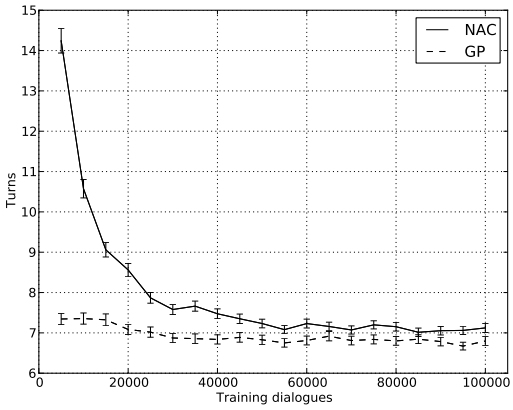

Figure 12: Comparison of the performance of GP-Sarsa vs the natural actor critic algorithm, where the performance is measured as the average number of turns as a function of the number of training dialogues.

in terms of dialogue length which supports the intuition that parametric policy modelling limits the optimality of the solution.

The fully trained GP and NAC policies were also compared on the simulated user by performing 1000 dialogues over a range of confusion rates. The average reward is given in Fig. 13 for both policies.

The results show that the GP policy outperforms the NAC policy across confusion rates in the neighbourhood of 15% confusion rate where the policies are trained. This shows that while the GP has a potential for more robust performance, it is important that the training conditions match testing conditions.

Evaluating policy performance on the same simulated user as used for training may be misleading. Hence, the policies were also tested in a trial with real human users recruited

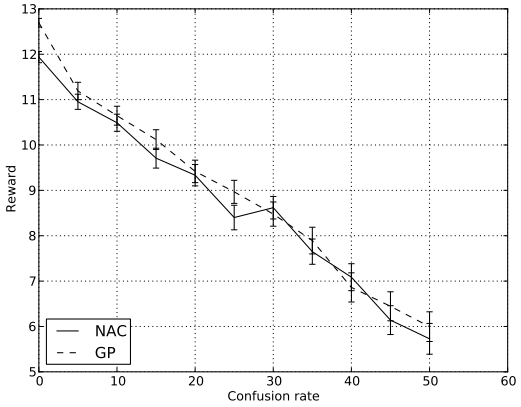

Figure 13: Comparison of the performance of GP-Sarsa vs the natural actor critic algorithm, where the performance is measured as the average reward as a function of the confusion rate in the user input.

using the Amazon Mechanical Turk service. Four policies were compared: partially trained GP and NAC policies, using only $10,000$ dialogues, and the fully trained GP and NAC policies.

Table 1: Human evaluation of simulator trained policies

	#of dialogues	Reward	Success	Turns
NAC-PartlyTrained	402	8.9 ± 0.4	87.3 ± 1.7	8.5 ± 0.2
NAC-FullyTrained	415	11.9 ± 0.3	91.8 ± 1.3	6.5 ± 0.2
GP-PartlyTrained	400	12.5 ± 0.3	93.5 ± 1.2	6.2 ± 0.2
GP-FullyTrained	397	11.6 ± 0.4	91.2 ± 1.4	6.6 ± 0.2

The results are given in Table 1, where for each policy the number of dialogues is given together with the average reward, the success rate and the average number of turns as well as their respective one standard errors. The average word error rate was 19%. These results show that the partially trained GP policy outperforms the partially trained NAC policy. Indeed, the partially trained GP policy outperforms all of the other policies including the fully trained GP policy. The latter result may be due to over-fitting, perhaps by over-exploiting traits in the simulated user behaviour that are not normally present in real human interaction.

Real world application

GP-Sarsa has been previously used for training a dialogue manager in direct interaction with real users [49], however, the summary state space used in that experiment was only four dimensional. In addition, the learning process exhibited inconsistencies due to the users not being able to correctly rate the dialogue. Here we report on an on-line learning experiment with the Bayesian Update of Dialogue State manager using a refined reward function.

The GP-Sarsa algorithm was implemented in a live telephone-based spoken dialogue system based on the BUDS framework operating with the summary space described in

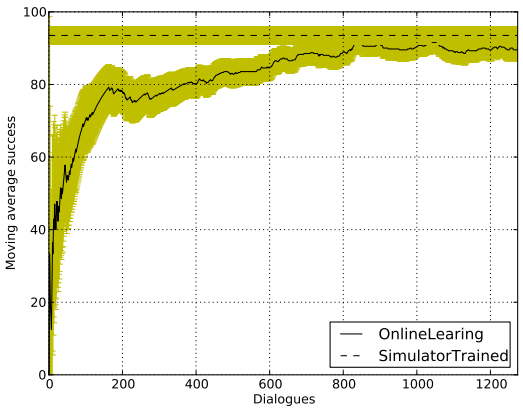

Figure 14: Comparison of the performance of the policy trained with the simulated user and tested with real people with the policy that is optimised online with interaction with real people. The performance is measured as the moving average success as a function of the number of dialogues.

Section 4.1. The GP-Sarsa configuration consisted of the Gaussian kernel ($\sigma_k = 5$ and $p = 4$) with a stochastic policy model where $\eta = 3$, the noise of the residual σ was set at 5 and the sparsification threshold ν was set at 0.1.

Human users were assigned specific tasks in the Cambridge restaurant domain using the Amazon Mechanical Turk service. At the end of each call, users were asked to press 1 if they were satisfied (i.e. believed that they had been successful in fulfilling the assigned task) and 0 otherwise. Previous experience suggests that this subjective user feedback is not sufficiently robust to rely on solely for the reward function. Therefore, at the end of each call the objective success was also calculated by comparing the predefined task with the system actions. The user rating was then used to compute the reward function only if it agreed with the objective measure of success. It has been shown previously that this can result in better policy performance [49].

The performance achieved during on-line learning on the initial 1274 dialogues was compared to the performance of the policy trained on 10,000 dialogues using the simulated user (GPPartlyTrained policy from the previous section). The results are given in Fig. 14 where the moving average success is given for a window of 400 dialogues along with the 95% confidence interval. Of the 1274 dialogues, 1081 had the same subjective and objective success ratings and hence only these 1081 dialogues were used for training. It can be seen from Fig. 14 that the policy trained on-line reaches a comparable asymptotic reward as the policy trained using the simulated user but in substantially fewer training dialogues. Moreover, the learning curve is smooth and it does not exhibit inconsistencies as was the case in earlier experiments [49].

Finally, the policy trained on 1000 dialogues with real users was tested under the same conditions as for the simulator trained policies (i.e. the scaling factor $\eta = 1$). Comparing Table 2 with Table 1, it can be seen that the policy trained on real users significantly outperforms the simulator trained policies both in terms of success rate and average reward.

Table 2: Human evaluation of the policy trained online

	#of dialogues	Reward	Success	Turns
GP-OnlineTrained	410	13.4 ± 0.3	96.8 ± 0.9	6.0 ± 0.1

4.4 Policy optimisation on the full belief space

The second major benefit of using Gaussian process policy optimisation is its potential to substantially reduce designer effort in defining the policy representation by avoiding the need to construct *ad hoc* feature mapping functions. We have already seen from the previous section that the GP approach does not require a set of basis functions to construct a parametric policy representation. According to the theory described in earlier sections, it is possible to train the policy directly on the full belief space by defining an appropriate kernel function. In this section, the ability to optimise a policy on the full belief space is examined in practice.

The kernel function on the full belief space

In the case of the full belief space, the kernel function must be defined over distributions. For the experiments described here, the kernel functions were defined following the approaches described in [51, 52] but modified to suit the spoken dialogue system application. The kernel function over the belief state was constructed from the sum of individual kernels over the distributions $\mathbf{b}_k$ for each hidden node k in the Bayesian network belief state $\mathbf{b}$.

We examined three kernel functions. The first is the expected likelihood kernel, which is also a simple linear inner product:

$$k_{\mathcal{B}}(\mathbf{b}, \mathbf{b}') = \sum_k \langle \mathbf{b}_k, \mathbf{b}'_k \rangle, \tag{26}$$

It measures the expected value of distribution $\mathbf{b}_k$ under distribution $\mathbf{b}'_k$.

The second is the Bhattacharyya kernel

$$k_{\mathcal{B}}(\mathbf{b}, \mathbf{b}') = \sum_k \sum_{i=0}^{i<d_k} \sqrt{\mathbf{b}_k(i)} \sqrt{\mathbf{b}'_k(i)}, \tag{27}$$

where d_k is the dimensionality of the k concept in the Bayesian network. It is related to Hellinger's distance, which can be viewed as a symmetric approximation of the Kullback-Leibler divergence [51].

The third kernel that we compare is given by

$$\begin{aligned} k_{\mathcal{B}}(\mathbf{b}, \mathbf{b}') &= \tfrac{-1}{\log(2)} \textstyle\sum_k \sum_{i=0}^{i<d_k} \mathbf{b}_k(i) \log(\frac{\mathbf{b}_k(i)}{\mathbf{b}_k(i)+\mathbf{b}'_k(i)}) \\ &\quad + \mathbf{b}'_k(i) \log(\frac{\mathbf{b}'_k(i)}{\mathbf{b}_k(i)+\mathbf{b}'_k(i)}). \end{aligned} \tag{28}$$

This kernel is related to Jansen-Shannon divergence, which is a symmetric and smoothed variant of the Kullback-Leibler divergence [52].

The kernel functions of the history nodes are defined directly over their respective distributions. Whilst it is possible to calculate the kernel function for the goal nodes in the same way, in this case, the choice of system action, such as *confirm* or *inform*, does not depend on the distribution of actual values, rather it depends on the shape of the distribution and, in particular, on the probability of the most likely value compared to the rest. Therefore, the kernel functions for goal nodes are calculated over vectors, where each vector represents

the corresponding distribution sorted into order of probability. The only exceptions are the *method* goal node and the *discourse act* node. The former defines whether the user is searching for a venue *by name* or *by constraints* and the latter defines which discourse act the user used, eg. *acknowledgement*, *thank you*. Their kernels are calculated in the same way as for the history nodes.

The sparsification threshold ν for this space is set at 0.001. Some intermediate experimentation showed that the kernel over the full belief space requires a finer span, so the threshold here is low. Since the same reward function is used for both the summary space and the full belief space, the variance of the Q-function residual σ for the full belief space was also fixed at 5. The sampling scale η was set to 3 and the results were compared with the Gaussian kernel operating on the summary space.

Training and evaluation on simulated user

The GP-Sarsa stochastic policy (15) that operates on the summary space was compared to the two policies operating on the full belief space that use different kernel functions. All training procedures used exactly the same number of dialogues in interaction with the simulated user in the training schedule described in Section 4.1.

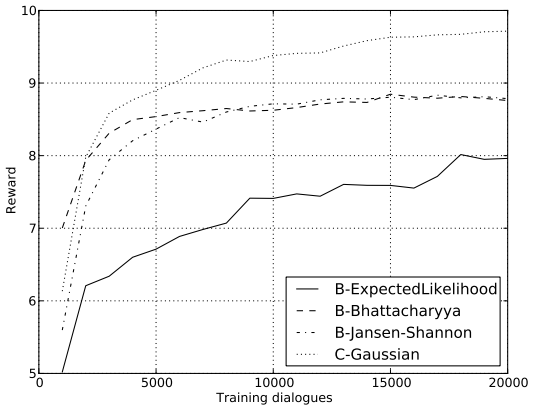

Figure 15: Comparison of the performance of 3 different policies operating on the full belief space ($\mathcal{B}$) with a policy operating on the summary space ($\mathcal{C}$).

The results are shown in Fig. 15. As can be seen, the GP policies trained on the full belief space do converge to an optimum, with the Bhattacharyya kernel performing best. Furthermore, increasing the dimensionality of the input space from the summary space to the full belief space did not slow down the process of learning. Overall performance is not however competitive with the summary space stochastic policy. This is disappointing but perhaps not surprising since defining a feature based summary space mapping can be viewed as designing a special kind of kernel that is hand-crafted to suit this task well. This is also the first attempt at designing a general-purpose kernel to fit an arbitrarily large belief space, and further work is clearly needed to refine this process.

5 Related work on GP-based RL

The methods and experiments for GP-based dialogue policy optimisation described in this paper have focussed on on-line and on-policy learning where the update takes place after

every dialogue turn (see Algorithm 1 in Section 3.2).[5] This approach is desirable in the case where a meaningful immediate reward can be obtained and when the dialogue takes many turns so it can be beneficial to perform the update during the dialogue. For example, in the future, it may be possible to configure an emotion detector to provide an immediate reward directly related to user satisfaction.

While on-policy learning has the desirable property of generating the highest reward during training [7], in practice it may sometimes be useful to perform off-policy learning. In that way one can guarantee a particular policy performance during on-line learning or learn directly off-line from a dialogue corpus.

Such an approach is taken in Gaussian process relational reinforcement learning [53], where off-policy learning takes place and the update is performed at the end of each episode. This approach does not necessitate that the relationship is established between the reward and the Q-function posterior as in (10), but instead it directly calculates the Q-value estimate $\hat{q}_t$:

$$\hat{q}^t = r_{t+1}(b_t = \mathbf{b}, a_t = a) + \gamma \max_{a'} \overline{Q}(\mathbf{b}', a'), \tag{29}$$

where $\overline{Q}$ is the current estimate of the Q-function mean. This then allows for the standard Gaussian process regression to be used to estimate the Q-function posterior given a sequence of the Q-values $\hat{\mathbf{q}}_t = [\hat{q}^1, \ldots, \hat{q}^t]^\mathsf{T}$ and the belief-action pairs $\mathbf{B}_t$ visited during a dialogue.

6 Conclusion

This paper has described how Gaussian processes can be applied successfully to POMDP-based dialogue management. Various aspects of Gaussian process reinforcement learning have been discussed, including sparse approximations, the use of a GP-based stochastic policy model and the GP-Sarsa optimisation algorithm.

Using a system based on the Bayesian Update of Dialogue State (BUDS) dialogue manager, various issues have been investigated experimentally both in simulation and with real users. The principal results are firstly that GP-based policy optimisation is faster than conventional gradient based methods and furthermore, performance in the early stages of learning becomes acceptable very quickly making it feasible to train systems from scratch directly in interaction with real users. Secondly, policies trained with real users significantly outperform policies trained on user simulators. Thirdly, although there is clearly more work to be done, it is feasible to construct a GP-based policy directly on the full POMDP belief space obviating the need for handcrafting a summary space mapping. These three results represent an important step towards building flexible dialogue systems that can range over large and varied domains dynamically adapting to the local context.

Future research in this area will need to address a number of issues. Firstly, improvements to kernel design are needed to enable effective kernels to be built for large continuous state spaces, this includes defining the kernel parameters and the Q-function residual noise σ (see Section 3.1). Better kernel approximation strategies are needed. For example, rather than approximating the Gram matrix as in (19), the Fully Independent Training Conditional approximation method [54, 38] may be more effective, where the Gram matrix is approximated as

$$\mathbf{K}_t \approx \mathbf{G}_t \tilde{\mathbf{K}}_t \mathbf{G}_t^\mathsf{T} - \mathrm{diag}(\mathbf{G}_t \tilde{\mathbf{K}}_t \mathbf{G}_t^\mathsf{T}) + \mathrm{diag}(\mathbf{K}_t). \tag{30}$$

With this approach, more variability would be allowed on the diagonal, which should reduce the problem of overly confident estimation. In addition, methods are needed for optimising the kernel function parameters. It has been shown on a toy dialogue problem that

[5] While the algorithm is designed to allow an update after every dialogue turn, in practice it is more convenient to apply the update after every dialogue.

these parameters can be learnt off-line directly from data by maximising the marginal likelihood (11) [35, 34]. This type of approach needs to be extended to work effectively in a full-scale real-world dialogue manager.

Secondly, the work presented here still requires the use of a summary action space and heuristic mapping functions to generate actual system responses. For the experiments here, a simple δ-kernel was used as the kernel function over the summary action space. However, Gaussian process reinforcement learning should allow more elaborate kernel functions to be defined which can be applied directly to the full action space. Since the full action space is not finite, sampling methods would then need to be developed to select an appropriate system response as in [31].

Finally, more work is needed to develop effective training strategies which allow optimisation of GP-Sarsa policies operating on the full belief state space in direct interaction with real users. The preliminary experiments reported in this paper suggest this is possible, but work is clearly needed to achieve performance levels which are competitive with current summary space models. A related issue concerns the design of reward functions, and robust methods of computing them on-line. In all of the work described in this paper, a simple fixed reward for success is given at the end of the dialogue in conjunction with a per turn penalty and the primary measure of success is determined by asking the user to give feedback. This learning regime would not be possible in real applications. Ideally the reward needs to be computed without the conscious engagement of users and it needs to be refined to encourage the type of desirable dialogue behaviours that are embodied in the design codes of current hand-crafted spoken dialogue applications.

1 Gaussian process regression

A Gaussian process is a set of random variables, any finite subset of which is jointly Gaussian distributed. A Gaussian process, denoted as $f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x})\right)$, is fully specified by its mean function $m(\mathbf{x}) = E\left(f(\mathbf{x})\right)$ and its covariance function $k(\mathbf{x}, \mathbf{x}') = E\left((f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))\right)$, also called the kernel function. The kernel function is a positive definite function that represents the function correlations. It defines how the function values in two points are correlated. The positive definiteness ensures that any multivariate Gaussian distribution derived from the Gaussian process is well-defined.

When using a Gaussian process, one starts by specifying the prior and the aim is then to compute a posterior given some observations. In the simplest case of GP regression, $f(\mathbf{x})$ is the unknown function for which an approximation is sought, and a zero-mean Gaussian process with kernel function $k(\mathbf{x}, \mathbf{x}')$ is assumed as the prior distribution, $f(\mathbf{x}) \sim \mathcal{GP}\left(0, k(\mathbf{x}, \mathbf{x})\right)$. The posterior distribution of $f(\mathbf{x})$ given the function values at some representative data points could then be computed using simple algebraic formulae [47].

In most real-world applications however, the true value of the function at the data points is not available, and only some noisy observations may be obtained. Let $\mathbf{y}_t = [y_1, \ldots, y_t]^\mathsf{T}$ be noisy observations of the function at data points $\mathbf{X}_t = [\mathbf{x}_1, \ldots, \mathbf{x}_T]^\mathsf{T}$, and $y_i = f(\mathbf{x}_i) + \xi$ is assumed, where the noise is additive, independent and Gaussian distributed, $\xi \sim \mathcal{N}(0, \sigma^2)$. For a Gaussian process, the joint distribution of the observed values and the function value at a test point, $f(\mathbf{x})$, can be shown to be [47]:

$$\begin{bmatrix} \mathbf{y}_t \\ f(\mathbf{x}) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}_t + \sigma^2 I & \mathbf{k}_t(\mathbf{x}) \\ \mathbf{k}_t^\mathsf{T}(\mathbf{x}) & k(\mathbf{x}, \mathbf{x}) \end{bmatrix}\right), \tag{31}$$

where $\mathbf{K}_t$ is the Gram matrix, *i.e.,* the matrix of kernel function values between each pair of data points, $(\mathbf{K}_t)_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, where $i, j \in \{1, \ldots, t\}$, and $\mathbf{k}_t(\mathbf{x})$ is the vector of kernel function values between test point $\mathbf{x}$ and each data point, $\mathbf{k}_t(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1) \ldots k(\mathbf{x}, \mathbf{x}_t)]^\mathsf{T}$.

Conditioning the joint Gaussian prior (31) on the observations yields the posterior of $f(\mathbf{x})$ [47]:

$$\begin{aligned} f(\mathbf{x})|\mathbf{y}_t &\sim \mathcal{N}\left(\overline{f}(\mathbf{x}), \mathrm{cov}(\mathbf{x},\mathbf{x})\right), \\ \overline{f}(\mathbf{x}) &= \mathbf{k}_t(\mathbf{x})^\mathsf{T}(\mathbf{K}_t + \sigma^2 I)^{-1}\mathbf{y}_t, \\ \mathrm{cov}(\mathbf{x},\mathbf{x}) &= k(\mathbf{x},\mathbf{x}) - \mathbf{k}_t(\mathbf{x})^\mathsf{T}(\mathbf{K}_t + \sigma^2 I)^{-1}\mathbf{k}_t(\mathbf{x}). \end{aligned} \tag{32}$$

From (32) it is required that data points $\mathbf{X}_t$ are known since $\mathbf{y}_t$ are observations at these points. In the literature this is often emphasised by the notation $f(\mathbf{x})|\mathbf{y}_t, \mathbf{X}_t$. From that notation, it is clear that the posterior depends on data points and the observations at the data points. It is important to note, however, that the variance of the posterior only depends on the data points and not on the observations. With more observations, the variance decreases and the estimate of the mean is refined.

2 Episodic GP-Sarsa algorithm

For a set of visited belief state-action pairs $\mathbf{B}_t = [(\mathbf{b}^0, a^0), \ldots, (\mathbf{b}^t, a^t)]^\mathsf{T}$, observed immediate rewards $\mathbf{r}_t = [r^1, \ldots, r^t]^\mathsf{T}$ and the dictionary of m representative points $\mathcal{D}_t = \{(\tilde{\mathbf{b}}^0, \tilde{a}^0), \ldots, (\tilde{\mathbf{b}}^m, \tilde{a}^m)\}$, the posterior of the Q-function, repeated from (20), can be written as:

$$Q(\mathbf{b},a)|\mathbf{B}_t, \mathbf{r}_t \sim \tag{33a}$$

$$\mathcal{N}(\tilde{\mathbf{k}}_t(\mathbf{b},a)^\mathsf{T}\tilde{\boldsymbol{\mu}}_t, k((\mathbf{b},a),(\mathbf{b},a)) - \tilde{\mathbf{k}}_t(\mathbf{b},a)^\mathsf{T}\tilde{\mathbf{C}}_t\tilde{\mathbf{k}}_t(\mathbf{b},a)), \tag{33b}$$

$$\tilde{\boldsymbol{\mu}}_t = \tilde{\mathbf{H}}_t^\mathsf{T}\tilde{\mathbf{W}}_t\mathbf{r}_t, \tag{33c}$$

$$\tilde{\mathbf{C}}_t = \tilde{\mathbf{H}}_t^\mathsf{T}\tilde{\mathbf{W}}_t\tilde{\mathbf{H}}_t, \tag{33d}$$

$$\tilde{\mathbf{W}}_t = (\tilde{\mathbf{H}}_t\tilde{\mathbf{K}}_t\tilde{\mathbf{H}}_t^\mathsf{T} + \sigma^2\tilde{\mathbf{H}}_t\tilde{\mathbf{H}}_t^\mathsf{T})^{-1}, \tag{33e}$$

$$\tilde{\mathbf{H}}_t = \mathbf{H}_t\mathbf{G}_t, \tag{33f}$$

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & -\gamma \end{bmatrix}, \tag{33g}$$

$$\mathbf{G}_t = [(\tilde{\mathbf{K}}_t^{-1}\tilde{\mathbf{k}}_t(\mathbf{b}^0,a^0))^\mathsf{T}, \ldots, (\tilde{\mathbf{K}}_t^{-1}\tilde{\mathbf{k}}_t(\mathbf{b}^t,a^t))^\mathsf{T}]^\mathsf{T}, \tag{33h}$$

and

$$\tilde{\mathbf{k}}_t(\mathbf{b},a) = [k((\mathbf{b},a),(\tilde{\mathbf{b}}^0,\tilde{a}^0)), \ldots, k((\mathbf{b},a),(\tilde{\mathbf{b}}^m,\tilde{a}^m))]^\mathsf{T}, \tag{33i}$$

$$\tilde{\mathbf{K}} = [\tilde{\mathbf{k}}_t(\tilde{\mathbf{b}}^0,\tilde{a}^0)^\mathsf{T}, \ldots, \tilde{\mathbf{k}}_t(\tilde{\mathbf{b}}^m,\tilde{a}^m)^\mathsf{T}]^\mathsf{T}. \tag{33j}$$

For two consequent belief state-action pairs and an associated reward ($\mathbf{b}^t$, a^t, r^{t+1}, $\mathbf{b}^{t+1}$, a^{t+1}), the posterior $Q(\mathbf{b},a)|\mathbf{B}_{t+1}, \mathbf{r}_{t+1}$ needs to be calculated efficiently, where $\mathbf{B}_{t+1} = [\mathbf{B}_t^\mathsf{T}, (\mathbf{b}^{t+1}, a^{t+1})]^\mathsf{T}$ and $\mathbf{r}_{t+1} = [\mathbf{r}_t^\mathsf{T}, r^{t+1}]^\mathsf{T}$. In [39] the sufficient statistics that enable this with complexity $O(m^2)$, where m is the size of the dictionary, are given for non-episodic tasks. Here, this is extended to support episodic tasks.

The main difference between episodic and non-episodic tasks is in the way the Q-function and the reward are related.

Algorithm 16 gives a full description of GP-Sarsa for episodic tasks. It recalculates the sufficient statistics of the Gaussian process model for the Q-function after every turn. The algorithm starts by initialising belief state $\mathbf{b}$. At the beginning of first episode action

Figure 16: Episodic GP-Sarsa

1: Initialise $\tilde{\boldsymbol{\mu}} \leftarrow []$, $\tilde{\mathbf{C}} \leftarrow []$, $\tilde{\mathbf{c}} \leftarrow []$, $d \leftarrow 0$, $1/v \leftarrow 0$
2: **for** each episode **do**
3: Initialise $\mathbf{b}$
4: **if** first episode **then**
5: Choose a arbitrary, $\mathcal{D} = \{(\mathbf{b}, a)\}$, $\tilde{\mathbf{K}}^{-1} \leftarrow 1/k((\mathbf{b}, a), (\mathbf{b}, a))$
6: **else**
7: **if** initial step **then**
8: Choose $a \leftarrow \pi(\mathbf{b})$
9: **end if**
10: **end if**
11: $\tilde{\mathbf{c}} \leftarrow \mathbf{0}$ {size $|\mathcal{D}|$}, $d \leftarrow 0$, $1/v \leftarrow 0$
12: $\mathbf{g} \leftarrow \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}}(\mathbf{b}, a)$, $\delta \leftarrow k((\mathbf{b}, a), (\mathbf{b}, a)) - \tilde{\mathbf{k}}(\mathbf{b}, a)^{\mathsf{T}}\mathbf{g}$
13: **if** $\delta > \nu$ **then**
14: $\mathcal{D} \leftarrow \{(\mathbf{b}, a)\} \cup \mathcal{D}$, $\tilde{\mathbf{K}}^{-1} \leftarrow \frac{1}{\delta}\begin{bmatrix} \delta\tilde{\mathbf{K}}^{-1} + \mathbf{g}\mathbf{g}^{\mathsf{T}} & -\mathbf{g} \\ -\mathbf{g}^{\mathsf{T}} & 1 \end{bmatrix}$
15: $\mathbf{g} \leftarrow [0, \ldots, 0, 1]^{\mathsf{T}}$ {size $|\mathcal{D}|$}, $\tilde{\boldsymbol{\mu}} \leftarrow \begin{bmatrix} \tilde{\boldsymbol{\mu}} \\ 0 \end{bmatrix}$, $\tilde{\mathbf{C}} \leftarrow \begin{bmatrix} \tilde{\mathbf{C}} & \mathbf{0} \\ \mathbf{0}^{\mathsf{T}} & 0 \end{bmatrix}$, $\tilde{\mathbf{c}} \leftarrow \begin{bmatrix} \tilde{\mathbf{c}} \\ 0 \end{bmatrix}$
16: **end if**
17: **for** each step in the episode **do**
18: Take a, observe r', update $\mathbf{b}'$
19: **if** non-terminal step **then**
20: Choose new action $a' \leftarrow \pi(\mathbf{b}')$
21: $\mathbf{g}' \leftarrow \tilde{\mathbf{K}}^{-1}\tilde{\mathbf{k}}(\mathbf{b}', a')$, $\delta \leftarrow k((\mathbf{b}', a'), (\mathbf{b}', a')) - \tilde{\mathbf{k}}(\mathbf{b}', a')^{\mathsf{T}}\mathbf{g}'$
22: $\Delta\tilde{\mathbf{k}} \leftarrow \tilde{\mathbf{k}}(\mathbf{b}, a) - \gamma\tilde{\mathbf{k}}(\mathbf{b}', a')$
23: **else**
24: $\mathbf{g}' \leftarrow [\mathbf{0}]$ {size $|\mathcal{D}|$}, $\delta \leftarrow 0$, $\Delta\tilde{\mathbf{k}} \leftarrow \tilde{\mathbf{k}}(\mathbf{b}, a)$
25: **end if**
26: $d \leftarrow \frac{\gamma\sigma^2}{v}d + r' - \Delta\tilde{\mathbf{k}}^{\mathsf{T}}\tilde{\boldsymbol{\mu}}$
27: **if** $\delta > \nu$ **and** non-terminal step **then**
28: $\mathcal{D} \leftarrow \{(\mathbf{b}', a')\} \cup \mathcal{D}$, $\tilde{\mathbf{K}}^{-1} \leftarrow \frac{1}{\delta}\begin{bmatrix} \delta\tilde{\mathbf{K}}^{-1} + \mathbf{g}\mathbf{g}^{\mathsf{T}} & -\mathbf{g} \\ -\mathbf{g}^{\mathsf{T}} & 1 \end{bmatrix}$
29: $\mathbf{g}' \leftarrow [0, \ldots, 0, 1]^{\mathsf{T}}$, $\tilde{\mathbf{h}} \leftarrow [\mathbf{g}^{\mathbf{T}}, -\gamma]^{\mathsf{T}}$
30: $\Delta k_{tt} \leftarrow \mathbf{g}^{\mathsf{T}}(\tilde{\mathbf{k}}(\mathbf{b}, a) - 2\gamma\tilde{\mathbf{k}}(\mathbf{b}', a')) + \gamma^2 k((\mathbf{b}', a'), (\mathbf{b}', a'))$
31: $\tilde{\mathbf{c}}' \leftarrow \frac{\gamma\sigma^2}{v}\begin{bmatrix} \tilde{\mathbf{c}} \\ 0 \end{bmatrix} + \tilde{\mathbf{h}} - \begin{bmatrix} \tilde{\mathbf{C}}\Delta\tilde{\mathbf{k}} \\ 0 \end{bmatrix}$
32: $v \leftarrow (1 + \gamma^2)\sigma^2 + \Delta k_{tt} - \Delta\tilde{\mathbf{k}}^{\mathsf{T}}\tilde{\mathbf{C}}\Delta\tilde{\mathbf{k}} + \frac{2\gamma\sigma^2}{v}\tilde{\mathbf{c}}\Delta\tilde{\mathbf{k}} - \frac{\gamma^2\sigma^4}{v}$
33: $\tilde{\boldsymbol{\mu}} \leftarrow \begin{bmatrix} \tilde{\boldsymbol{\mu}} \\ 0 \end{bmatrix}$, $\tilde{\mathbf{C}} \leftarrow \begin{bmatrix} \tilde{\mathbf{C}} & \mathbf{0} \\ \mathbf{0}^{\mathsf{T}} & 0 \end{bmatrix}$
34: **else**
35: $\check{\mathbf{h}} \leftarrow \mathbf{g} - \gamma\mathbf{g}'$, $\tilde{\mathbf{c}}' \leftarrow \frac{\gamma\sigma^2}{v}\tilde{\mathbf{c}} + \check{\mathbf{h}} - \tilde{\mathbf{C}}\Delta\tilde{\mathbf{k}}$
36: **if** non-terminal step **then**
37: $v \leftarrow (1 + \gamma^2)\sigma^2 + \Delta\tilde{\mathbf{k}}^{\mathsf{T}}(\tilde{\mathbf{c}}' + \frac{\gamma\sigma^2}{v}\tilde{\mathbf{c}}) - \frac{\gamma^2\sigma^4}{v}$
38: **else**
39: $v \leftarrow \sigma^2 + \Delta\tilde{\mathbf{k}}^{\mathsf{T}}(\tilde{\mathbf{c}}' + \frac{\gamma\sigma^2}{v}\tilde{\mathbf{c}}) - \frac{\gamma^2\sigma^4}{v}$
40: **end if**
41: **end if**
42: $\tilde{\boldsymbol{\mu}} \leftarrow \tilde{\boldsymbol{\mu}} + \frac{\tilde{\mathbf{c}}}{v}d$, $\tilde{\mathbf{C}} \leftarrow \tilde{\mathbf{C}} + \frac{1}{v}\tilde{\mathbf{c}}'\tilde{\mathbf{c}}'^{\mathsf{T}}$, $\tilde{\mathbf{c}} \leftarrow \tilde{\mathbf{c}}'$, $\mathbf{g} \leftarrow \mathbf{g}'$
43: **if** non-terminal step **then**
44: $\mathbf{b} \leftarrow \mathbf{b}'$, $a \leftarrow a'$
45: **end if**
46: **end for**
47: **end for**

a is taken randomly, the dictionary is initialised and the inverse Gram matrix trivially calculated (line 5). Otherwise action a is taken ϵ-greedily (13), using active learning (14) or the stochastic policy (15) with respect to the current estimate of $\tilde{\boldsymbol{\mu}}$ (line 7) which determines the mean of the Q-value (33b). The sufficient statistics are then copied from the end of the previous episode (line 9). If the sparsification threshold is exceeded the dictionary is expanded and the sufficient statistics are recalculated (lines 10-14).

For each step in the episode (line 16) action a is taken, reward r' observed, belief state updated $\mathbf{b}'$ and next action a' chosen ϵ-greedily (13), using active learning (14) or the stochastic policy (15). In the final step of the episode, the system takes action a in the penultimate belief state $\mathbf{b}$, obtains the reward r' and then moves to the final belief state $\mathbf{b}'$. The next belief state $\mathbf{b}'$ and its residual are only taken into consideration when the current time step is not final, see lines 17-20 and lines 33-35 respectively. Also, only non-final belief states are included in the dictionary (lines 24-30).

Acknowledgments

The authors would like to thank Blaise Thomson, who built the Bayesian Update of Dialogue State system used here, Jost Schatzmann and Simon Keizer, who built the user simulator and Pirros Tsiakoulis for his help with the user trials. The authors would also like to thank Matt Henderson, Kai Yu, Martin Szummer, Catherine Breslin and Dongho Kim for useful comments and discussions.

References

[1] M. McTear, "Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit," in *Proceedings of ICSLP*, 1998.

[2] R. Pieraccini and J. Huerta, "Where do we go from here? Research and commercial spoken dialog systems," in *Proceedings of SIGDIAL*, 2005.

[3] S. Young, "Talking to Machines (Statistically Speaking)," in *Proceedings of ICSLP*, 2002.

[4] E. Levin, R. Pieraccini, and W. Eckert, "Using Markov Decision Processes for Learning Dialogue Strategies," in *Proceedings of ICASSP*, 1998.

[5] S. Singh, M. Kearns, D. Litman, and M. Walker, "Reinforcement Learning for Spoken Dialogue System," in *Proceedings of NIPS*, 1999.

[6] E. Levin, R. Pieraccini, and W. Eckert, "A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 1, pp. 11–23, 2000.

[7] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning, Cambridge, Massachusetts: MIT Press, 1998.

[8] S. Young, "Probabilistic Methods in Spoken Dialogue Systems," *Philosophical Trans Royal Society (Series A)*, vol. 358, no. 1769, pp. 1389–1402, 2000.

[9] D. J. Litman, M. S. Kearns, S. Singh, and M. A. Walker, "Automatic optimization of dialogue management," in *Proceedings of ACL*, 2000.

[10] D. Goddeau and J. Pineau, "Fast reinforcement learning of dialog strategies," in *Proceedings of ICASSP*, 2000.

[11] S. Singh, D. Litman, M. Kearns, and M. Walker, "Optimizing Dialogue Management with Reinforcement Learning," *Journal of Artificial Intelligence Research*, vol. 16, pp. 105–133, 2002.

[12] K. Georgila and O. Lemon, "Adaptive multimodal dialogue management based on the information state update approach," in *W3C Workshop on Multimodal Interaction*, 2004.

[13] N. Roy, J. Pineau, and S. Thrun, "Spoken dialogue management using probabilistic reasoning," in *Proceedings of ACL*, 2000.

[14] B. Zhang, Q. Cai, J. Mao, E. Chang, and B. Guo, "Spoken Dialogue Management as Planning and Acting under Uncertainty," in *Proceedings of Eurospeech*, 2001.

[15] J. Williams and S. Young, "Partially Observable Markov Decision Processes for Spoken Dialog Systems," *Computer Speech and Language*, vol. 21, no. 2, pp. 393–422, 2007.

[16] B. Thomson, *Statistical methods for spoken dialogue management*. PhD thesis, University of Cambridge, 2009.

[17] S. Young, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu, "The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management," *Computer Speech and Language*, vol. 24, no. 2, pp. 150–174, 2010.

[18] M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, K. Yu, and S. Young, "Training and evaluation of the HIS-POMDP dialogue system in noise," in *Proceedings of SIGDIAL*, 2008.

[19] F. Jurčíček, B. Thomson, and S. Young, "Natural actor and belief critic: Reinforcement algorithm for learning parameters of dialogue systems modelled as POMDPs," *ACM Transactions on Speech and Language Processing*, 2011.

[20] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[21] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proceedings of IJCAI*, pp. 1025–1032, 2003.

[22] J. Williams and S. Young, "Scaling POMDPs for Spoken Dialog Management," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 7, pp. 2116–2129, 2007.

[23] J. Henderson, O. Lemon, and K. Georgila, "Hybrid Reinforcement/Supervised Learning for Dialogue Policies from fixed data sets," *Computational Linguistics*, vol. 34, no. 4, pp. 487–511, 2008.

[24] B. Thomson and S. Young, "Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems," *Computer Speech and Language*, vol. 24, no. 4, pp. 562–588, 2010.

[25] L. Li, J. Williams, and S. Balakrishnan, "Reinforcement Learning for Dialog Management using Least-Squares Policy Iteration and Fast Feature Selection," in *Proceedings of Interspeech*, 2009.

[26] F. Pinault and F. Lefèvre, "Semantic graph clustering for POMDP-based spoken dialogue systems," in *Proceedings of Interspeech*, 2011.

[27] P. Crook and O. Lemon, "Lossless Value Directed Compression of Complex User Goal States for Statistical Spoken Dialogue Systems," in *Proceedings of Interspeech*, 2011.

[28] L. Daubigney, M. Geist, and O. Pietquin, "Off-policy Learning in Large-scale POMDP-based Dialogue Systems," in *Proceedings of ICASSP*, 2012.

[29] Amazon, "Amazon Mechanical Turk," 2011. `https://www.mturk.com/mturk/welcome`.

[30] Y. Engel, S. Mannor, and R. Meir, "Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning," in *Proceedings of ICML*, 2003.

[31] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *Proceedings of ICML*, 2005.

[32] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 16, pp. 751–759, MIT Press, 2004.

[33] M. Deisenroth, C. Rasmussen, and J. Peters, "Gaussian Process Dynamic Programming," *Neurocomputing*, vol. 72, no. 7-9, pp. 1508–1524, 2009.

[34] M. Gašić, *Statistical Dialogue Modelling*. PhD thesis, University of Cambridge, 2011.

[35] M. Gašić, F. Jurčíček, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, K. Yu, and S. Young, "Gaussian processes for fast policy optimisation of pomdp-based dialogue managers," in *Proceedings of SIGDIAL*, 2010.

[36] D. Cohn, L. Atlas, and R. Ladner, "Improving Generalization with Active Learning," *Machine Learning*, vol. 15, pp. 201–221, 1994.

[37] MacKay, D.J.C., "Information-based objective functions for active data selection," *Neural Computation*, vol. 4, no. 4, pp. 590–604, 1992.

[38] J. Quinonero-Candela and C. Rasmussen, "A Unifying View of Sparse Approximate Gaussian Process Regression," *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[39] Y. Engel, *Algorithms and Representations for Reinforcement Learning*. PhD thesis, Hebrew University, 2005.

[40] B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, 2001.

[41] M. Lázaro-Gredilla, J. Quiñonero Candela, C. Rasmussen, and A. Figueiras-Vidal, "Sparse Spectrum Gaussian Process Regression," *Journal of Machine Learning Research*, vol. 11, pp. 1865–1881, 2010.

[42] J. Peters and S. Schaal, "Natural Actor-Critic," *Neurocomputing*, vol. 71, pp. 1180–1190, 2008.

[43] J. Schatzmann, *Statistical User and Error Modelling for Spoken Dialogue Systems*. PhD thesis, University of Cambridge, 2008.

[44] S. Keizer, M. Gašić, F. Jurčíček, F. Mairesse, B. Thomson, K. Yu, and S. Young, "Parameter estimation for agenda-based user simulation," in *Proceedings of SIGDIAL*, 2010.

[45] B. Thomson, M. Gasic, M. Henderson, P. Tsiakoulis, and S. Young, "N-Best error simulation for training spoken dialogue systems," in *Proceedings of SLT*, 2012.

[46] F. Jurčíček, S. Keizer, M. Gašić, F. Mairesse, B. Thomson, K. Yu, and S. Young, "Real user evaluation of spoken dialogue systems using Amazon Mechanical Turk," in *Proceedings of Interspeech*, 2011.

[47] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, Massachusetts: MIT Press, 2005.

[48] L. Daubigney, M. Gašić, S. Chandramohan, M. Geist, O. Pietquin, and S. Young, "Uncertainty management for on-line optimisation of a POMDP-based large-scale spoken dialogue system," in *Proceedings of Interspeech*, 2011.

[49] M. Gašić, F. Jurčíček, B. Thomson, K. Yu, and S. Young, "On-line policy optimisation of spoken dialogue systems via live interaction with human subjects," in *Proceedings of ASRU*, 2011.

[50] H. Jakab and L. Csató, "Reinforcement learning with guided policy search using gaussian processes," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pp. 1–8, IEEE, 2012.

[51] T. Jebara, R. Kondor, and A. Howard, "Probability product kernels," *J. Mach. Learn. Res.*, vol. 5, pp. 819–844, Dec. 2004.

[52] M. Hein and O. Bousquet, "Hilbertian metrics and positive definite kernels on probability measures," in *Proceedings of AISTATS 2005*, 2004.

[53] K. Driessens, J. Ramon, and T. Gärtner, "Graph kernels and Gaussian processes for relational reinforcement learning," *Machine Learning*, vol. 64, no. 1, pp. 91–119, 2006.

[54] E. Snelson and Z. Ghahramani, "Sparse Gaussian Processes using Pseudo-inputs," in *Proceedings of NIPS*, 2006.