Training Dialogue Systems With Human Advice

Merwan Barlier
Université de Lille
SequeL Team

Romain Laroche
Microsoft Research
Montréal

Olivier Pietquin
Université de Lille
SequeL Team
Institut Universitaire de France, IUF

ABSTRACT

One major drawback of Reinforcement Learning (RL) Spoken Dialogue Systems is that they inherit from the general exploration requirements of RL which makes them hard to deploy from an industry perspective. On the other hand, industrial systems rely on human expertise and hand written rules so as to avoid irrelevant behavior to happen and maintain acceptable experience from the user point of view. In this paper, we attempt to bridge the gap between those two worlds by providing an easy way to incorporate all kinds of human expertise in the training phase of a Reinforcement Learning Dialogue System. Our approach, based on the TAMER framework, enables safe and efficient policy learning by combining the traditional Reinforcement Learning reward signal with an additional reward, encoding expert advice. Experimental results show that our method leads to substantial improvements over more traditional Reinforcement Learning methods.

KEYWORDS

Human-robot/agent interaction; Learning agent capabilities (agent models, communication, observation)

ACM Reference Format:
Merwan Barlier, Romain Laroche, and Olivier Pietquin. 2018. Training Dialogue Systems With Human Advice. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

Over the past few years, statistical methods have achieved several success in the dialogue domain: for dialogue simulation [9], natural language generation [49], generative dialogue systems [33]... The recent combination of Neural Networks with Reinforcement Learning for training goal-oriented dialogue systems led to great improvements in their capabilities and performance [12, 45]. Learning from raw data only, those models are attractive due to their efficiency and their generality, *i.e.* one architecture may address a wide variety of tasks with almost no human intervention.

While such systems have achieve many academic accomplishments, many challenges have yet to be overcome for their industrial release. First, data sparsity has to be efficiently managed. Deep learning is indeed known to need a lot of data to achieve its best performance. In those conditions, it is often impossible for a dialogue system to learn an accurate model of the conversation through only interactions with humans. Attempts have been made [8, 13] to make on-line learning of dialogue strategies as data-efficient as possible, but those methods have been largely surpassed in the meantime by Deep Learning algorithms [12], which do not solve the scalability issues arising when trying to learn an accurate model of their environment, *e.g.* the user of the dialogue system [30].

One way to address this problem is to first train a policy on a user simulator [3, 9, 43] together with error modeling [37, 48] in order to model the imperfections of the recognition and understanding modules, and then to bootstrap the learned strategy with real human interaction. This method presents the advantage of allowing the dialogue system to be trained on millions of dialogues before being released into the real world. However, the resulted strategy strongly depends on the quality of the simulated user and a perfect user model is not realizable in practice. For instance, in the beginning of the Reinforcement Learning phase, it is probable that the non conventional actions taken by the system will lead the user to unknown states, leading it to execute non accurate behaviors [42] and thus biasing the learning phase of the dialogue system.

Batch Reinforcement Learning [27, 29] provides an alternative to this on-line approach. The main idea behind this framework is to exploit the generalization performances of Approximate Dynamic Programming to get the most effective policy out of a set of transitions. This approach provides numerous advantages over learning a strategy by interactions. First, the batch of transitions is known and no assumptions have to be made on the way it has been collected. A different set of data may thus merge to provide a batch as exhaustive as possible. Second, Batch Reinforcement Learning is proven to be extremely data efficient, taking advantage of the generalization of supervised learning methods such as Deep Learning and needing no stochastic approximation techniques commonly used in on-line Reinforcement Learning. Finally, the asynchronous updates of the Q-function in on-line Reinforcement Learning with function approximation are known to cause stability issues [14]. Conversely, Batch Reinforcement Learning algorithms update all Q-values synchronously, and directly avoids those stability issues. For those reasons, Batch Reinforcement Learning methods such as LSPI [32] or Fitted-Q iteration [36] have been shown to be more effective than their on-line counterparts.

Another problem for the industrial release of statistical spoken dialogue systems is the need to ensure safe behaviors and ban dangerous or illegal actions. Quality-of-service is indeed in this case a key feature. A bad user experience, even only during the release of the product, may negatively affect its expansion. Human supervision is often the most adequate way to ensure this safety.

Many studies have focused on the integration of human guidance for Reinforcement Learning Spoken Dialogue System [4, 28, 51], or more generally any Reinforcement Learning agent [34]. Typically, these studies address the problem by constraining the actions available to the system in some states specified by a human, or make it follow some rules in those states. Once robust behaviors have been

Olivier Pietquin works now at Google Brain.
Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden. © 2018 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

learned through Reinforcement Learning, handcrafted behaviors are replaced bit by bit by learned ones.

An alternative to this approach is provided by Reward Shaping [35] and the TAMER+RL framework [24]. Both of those frameworks address the problem by encoding the supervision into an additional reward function. While the former acts by biasing the reward function during learning (and thus the Q-values) to match the expert recommendations, the latter directly acts on the policy of the learner. An advantage of those methods over the previous ones is that they provide a guarantee to converge towards a solution of the initial problem. They are thus in the end able to overcome the human bias that would lead to suboptimal behaviors [41]. However, contrary to TAMER+RL, Reward Shaping imposes to make strong assumptions on the form of the additional reward function to guarantee this optimal policy invariance. Providing the right additional reward function in those conditions is a difficult problem in itself.

However, all of the previous methods present the drawback of being designed to work with on-line Reinforcement Learning algorithms. This is why in this work, we introduce an alternative method based on the TAMER+RL framework and LSPI combining the data-efficiency and stability advantages of Batch Reinforcement Learning with the safety provided by expert supervision.

This paper is structured as follows. Section 2 recalls Reinforcement Learning preliminaries to introduce the framework. Section 3 provides the state of the art on designing reinforcement learning agents able to learn from advice and introduces TAMER-LSPI, our method to integrate human advice on a Batch Reinforcement Learning task. Finally the efficiency of our approach is outlined in Section 4 on an experimental protocol based on a simulated restaurant booking dialogue management task based on the DSTC 2 dataset.

2 PRELIMINARIES

2.1 Reinforcement Learning Dialogue Systems

Since [31, 44], automatic strategy learning for goal-oriented Spoken Dialogue Systems (SDS) is mainly addressed with the Reinforcement Learning (RL) framework [47]. Within this framework, one considers a user and a Dialogue Manager interacting through a noisy channel consisting of Automatic Speech Recognition (ASR) and Spoken Language Understanding (SLU).

Picking the right action, *i.e.* selecting the right thing to say, is cast as a Markov Decision Process (MDP) problem. Formally, an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, R, T, \gamma \rangle$ where $\mathcal{S}$ is the *state* space, $\mathcal{A}$ is the *action* space, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward* function, implicitly defining the task objective, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the *transition* function, representing the environment dynamics, and $\gamma \in [0, 1)$ is the *discount factor*. At each time step t, given a state $s \in \mathcal{S}$ called dialogue context, the dialogue manager takes an action $a \in \mathcal{A}$. Taking this action leads to another state $s' \in \mathcal{S}$ drawn according to the transition function T. While transitioning, the dialogue manager will also receive a scalar immediate reward r. The discounted return $\sum_t \gamma^t r_t$ assesses the success (or failure) of the dialogue.

The goal of the Reinforcement Learning based Dialogue Manager is to find a mapping $\pi : \mathcal{S} \to \mathcal{A}$, called *policy* that maximizes the expectation of the discounted returns, called *value*:

$$V^{\pi}(s) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \Big]. \quad (1)$$

Similarly, one defines another function, the Q-function as the value of taking an action in a given state:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \sum_{s' \in [S]} T(s, a, s')\pi(s), s')V(s'). \quad (2)$$

An order relation may then be defined on values and Q-functions: $Q_1 > Q_2$ if and only if $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}, Q_1(s, a) > Q_2(s, a)$. The optimal Q-function Q^* (resp. value V^*) is the Q-function maximal in each state. It may be proven that such a Q-function exists and is unique in every MDP. It is thus possible to define the optimal policy π^* such that: $\forall s \in \mathcal{S}$, $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$. For an on-line Reinforcement Learning agent, a common way to find the optimal policy is to approximate the optimal Q-function by repeatedly sampling through interactions with its environment, *i.e.* a user (which may be either real or simulated) in a dialogue domain.

In the Spoken Dialogue System case, the state space is the set of all possible dialogue contexts and the action space is the set of all dialogue acts the system is able to utter. In the case of task-oriented Spoken Dialogue system, the reward function is often a binary function assessing the success of the dialogue.

2.2 Least Square Policy Iteration

2.2.1 Batch Reinforcement Learning. The Batch Reinforcement Learning problem may be defined as the task of learning the best policy from a set of transitions collected a priori. In that setting, the learner is not allowed to interact with its environment. As described on figure 1, the Batch Reinforcement Learning setting consists of three steps. First, a set of transition $\mathcal{D} = [(s_i, a_i, s'_i, r_i)]_{i=1}^{N}$ is gathered. Since Batch Reinforcement Learning algorithms are off-policy algorithms, no assumption has to be made on the policy collecting the data. In the case of Spoken Dialogue Systems, to ensure that the transition function sampling the transitions will be similar to the one which will be used in practice, data may be gathered by a Wizard-of-Oz [52]. Second, a Batch Reinforcement Learning algorithm is computed over the set of transitions to derive the most effective policy out of it. Finally, the Reinforcement Learning agent applies the policy derived during the previous step into the real world.

This task is extremely convenient in an industrial context. Letting indeed a dialogue agent to take random, non-optimal actions with customers badly affects the quality of the service provided by the company. It is therefore often preferable to collect a large amount of data with a *safe* policy and then, punctually, once that the learned policy has been correctly evaluated, to update and release it.

2.2.2 Policy Iteration. The Policy Iteration algorithm is an algorithm from the Dynamic Programming literature which relies at the heart of a whole branch of Batch Reinforcement Learning algorithms. To use this algorithm, one needs first to assume that the MDP is known (*i.e.* the reward and transition functions are known). The Policy Iteration algorithm is then based on the following theorem ([39], p.175):

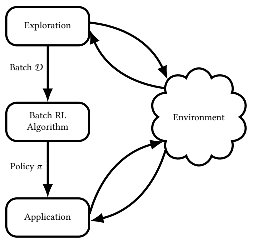

Figure 1: The three steps of Batch Reinforcement Learning : 1. Data Gathering with exploration 2. Application of a Batch Reinforcement Learning Algorithm 3. Application of the policy in the real world.

Theorem 2.1. *Given the value function V^{π_1} of some policy π_1, every policy π_2 such that $\forall s \in \mathcal{S}, \pi_2(s) \in \operatorname{argmax}_{\pi'}\{r(\pi'(s) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi'(s), s')V^{\pi_1}(s)\}$ (i.e every policy greedy with respect to that value function) has a greater (or equal) value function V^{π_2}. If $V^{\pi_1} = V^{\pi_2}$, then π_1 is the optimal policy π^*.*

Proof. $r^{\pi_2} + \gamma T^{\pi_2} V^{pi_1} \geq V^{\pi_1}$ (in matrix form). Rearranging the terms, $r_2^{\pi} >= (I - \gamma T^p i_2)V_1^{\pi}$. The result is then obtained by multiplying by $(I - \gamma T_2^{\pi})^{-1}$ and by taking advantage of the fact that by definition, $V^{\pi_2} = r^{\pi_2} + \gamma T^{\pi_2} r^{\pi_2} + \gamma^2 T^{\pi_2 2} r_2^{\pi} + \ldots = (I - \gamma T^{\pi_2})^{-1} r^{\pi_2}$ □

Thanks to this result, it is guaranteed that iteratively repeating a sequence of policy evaluation and policy improvement (by acting greedily with respect to that policy) leads to a sequence of improving policies converging towards the optimal one. The Policy Iteration algorithm is thus described by Algorithm 1:

```
Input  : An initial policy $\pi_0$
Output : The optimal policy $\pi^*$
repeat
    1. Compute the value $Q^{\pi_n}$ of $\pi_n$
    2. Set $\forall s \in \mathcal{S},\ \pi_{n+1}(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^{\pi_n}(s, a)$
until $\pi_{n+1} = \pi_n$;
```
Algorithm 1: Policy Iteration algorithm

2.2.3 *Least Square Policy Iteration.*

Least-Square Policy Iteration (LSPI, [26]) is an algorithm designed to solve the Batch Reinforcement Learning problem. It has been successfully applied for dialogue systems [32]. First, it is assumed that from each state, one can extract a set of n features characterizing this state. At each state s there is a corresponding feature vector $\Phi(s) = [\Phi_1(s), \ldots, \Phi_n(s)]$. LSPI also assumes that given a policy, the Q-function may be expressed as the inner product between these features and a set a of parameters θ to will be optimized by the algorithm: $Q^{\pi}(s) = \langle \Phi(s), \theta \rangle$. This assumption is common for Reinforcement Learning with large or continuous state spaces. In section 4, we will present a way to extract features from dialogue contexts.

LSPI follows the same scheme of policy evaluation/policy improvement that Policy Iteration. However, Since reward and transition functions are not known in the Batch Reinforcement Learning setting, one needs to estimate the Q-function during the evaluation step. As it has already been assumed that the Q-function could be represented as a linear combination of the state-action features, its estimations will be computed with linear regressions. It is proven [26] that the parameter θ^{π} of the Q-function evaluating the policy π, $Q^{\pi}(s) = \langle \Phi, \theta^{\pi} \rangle$ is the solution of the linear equation :

$$(A + \beta I)\theta^{\pi_n} = b, \tag{3}$$

where π_n is the policy being evaluated at the n^{th} policy iteration, where the ℓ_2 regularization parameter β has been added, as suggested in [25] to avoid overfitting issues. Here, I the identity matrix, and A and b are computed from the samples:

$$A = \sum_{i=1}^{N} \Phi(s_i, a_i)(\Phi(s_i, a_i) - \gamma \Phi(s_i', \pi_n(s_i')))^T$$

$$b = \sum_{i=1}^{N} \Phi(s_i, a_i) r_i$$

LSPI is thus described by Algorithm 2.

```
Input  : An initial policy $\pi_0$,
         A batch $\mathcal{D} = [(s_i, a_i, s_i', r_i)]_{i=1}^{N}$
Output : The optimal policy $\pi^*$
repeat
    1. Estimates $\hat{Q}^{\pi_n} = \langle \theta_n, \Phi \rangle$ where $\theta_n$ is the solution of the
       equation $(A + \beta I)\theta_n = b$ where $A$ and $b$ are computed
       from the samples:
       $A = \sum_{i=1}^{N} \Phi(s_i, a_i)(\Phi(s_i, a_i) - \gamma \Phi(s_i', \pi_n(s_i')))^T$,
       $b = \sum_{i=1}^{N} \Phi(s_i, a_i) r_i$ , $\beta$ is a regularizing constant and $I$
       the identity matrix
    2.Set $\forall s \in \mathcal{S},\ \pi_{n+1}(s) = \operatorname{argmax}_{a \in \mathcal{A}} \hat{Q}^{\pi_n}(s, a)$
until $\pi_{n+1} = \pi_n$;
```
Algorithm 2: LSPI algorithm

It is important to notice that the LSPI algorithm is a Policy Iteration-based algorithm, and so it is easy to construct variations of this algorithm to incorporate policy changes, as long as an improvement is made during the policy improvement step. As we will see in the next section, this is a key feature in our method for constructing agents receiving advice.

3 ADVISING AGENTS

3.1 State Of The Art

The problem of integrating advice in order to speed up learning for Reinforcement Learning agents and to provide safe behaviors has been addressed for a long time. First frameworks consisted in forgetting what has previously been learned in some states with Reinforcement Learning and instead applying a safe action provided by either the expert [7] or a set of rules, like for instance the RATLE algorithm [34]. Following this line of work, [28, 40] propose a modification of this process by first handcrafting a general high-level safe strategy, which is optimized at a finer grain on-line with Reinforcement Learning when enough data has been gathered.

An alternative way to mix safe handcrafted rules with robust ones learned with Reinforcement Learning is provided by [51]. In this work, the authors suggest to use an on-line learning mechanism which first selects a set of safe actions with handcrafted rules and selects among those actions with Reinforcement Learning.

Reward shaping [35] provides an alternative way to incorporate advice for speeding up the learning of an RL agent with a more subtle approach. Within this framework, an additional human-designed reward F is added to the traditional reward r. The idea behind this second reward is that sometimes, propagating the reward signal through the whole MDP may take a large number of time steps. The agent may have therefore difficulties to distinguish which actions lead to the success of the task or its failure. To overcome this, the human-designed reward may explicitly encourage the agent to take good actions and avoid bad ones. However, [35] prove that a sufficient and necessary condition to have a final optimal policy identical to the optimal policy in the MDP without the shaping reward, one needs to derive the shaping reward from state potentials, *i.e.* $F(s, s') = \gamma\Phi(s') - \Phi(s)$. [50] extended this result by showing that it was also possible to design a shaping function from state-action potentials. Deriving the shaping function from potentials is however a strong limitation of reward shaping. A Reinforcement Learning agent takes indeed its actions with respect to the Q-function. The shaping reward must thus respect the potential property while leading during learning to a Q-function corresponding to the human advice. [16] showed that designing such a function leads to a problem as hard as the Reinforcement Learning one. This paradigm has been applied on a variety of dialogue problems. For instance [10] speed-up the learning phase of a dialogue manager by using returns as shaping potentials, while [46] use returns derived from a dialogue simulator to speed-up learning while interacting with real users.

An alternative approach to incorporate advice while training a Reinforcement Learning agent is given by policy shaping [15]. In this work, the authors first learn a model of the advising human policy and use it to predict what should be his/her action in the situation of the agent. Then, this policy is mixed with the learned one in order to derive a final policy taking the advice into account. Deriving the policy of the expert from his/her advice is however a challenging problem in its own and to solve this problem, one has to restrict ourselves to very basic human advice models.

Recently, [4] introduced a method to incorporate direct demonstrations from experts to bias policy learning towards a safe and efficient human policy. In this work, authors suggest to pick the action provided by the expert when available. Otherwise, by introducing an additional reward signal, they incite the system to take the action the expert would have probably taken. However, this method suffers from the fact that the learner may not be able to overcome human bias. Furthermore, determining what may be the human action on unknown states is a difficult task [42].

The Reinforcement Learning with Expert Demonstrations problem [18, 22, 38] also addresses the task of adding expert information to the traditional reward signal. In that case, this additional information are demonstrations of optimal (or near-optimal) actions in some states. This problem is generally solved by constraining the learned Q-functions to have higher values for demonstrated samples. This framework may however not be applied in our case since, by definitions, demonstrations of the experts are optimal and it is not possible to incorporate "negative" feedbacks, *i.e.* telling the agent that its chosen action was not a particularly good one.

Finally, an alternative approach on the integration of human advice for Reinforcement Learning agents focus on ways to request assistance to a human expert when the agent encounters specific situations where it has not enough information to act properly. This problem was first addressed in [6] and answered with the `AskForHelp` system, where the Reinforcement Learning agent asks for help when given all the Q-values in a state are too similar too decide which action is the correct one. Following this line of work, [5] proposes to ask for a rollout from the expert when the agent is not confident about which action is safe.

3.2 The TAMER+RL framework

The TAMER (Training an Agent Manually via Evaluative Reinforcement) framework [23] was introduced in order to train agents directly from on-line human feedback, without "traditional" RL reward. A TAMER agent evolves as an RL agent in a environment given by a set of states and moves from states to states by taking actions. Like an RL agent, the TAMER agent receives a reward $H(s, a)$ after executing action a in state s. However, contrary to the RL agent, the reward is not previously handcrafted, it is instead directly given by the adviser. This leads to the main difference between the TAMER and RL frameworks, in the former case the agent does not try to optimize its cumulative rewards but acts greedily in order to maximize its immediate reward. The idea behind this paradigm is that when a human judges the action of an agent, s/he somehow considers and takes into account the future consequences of the action. The goal of the TAMER agent in a state s is first to correctly predict $H(s, a)$ for all actions $a \in \mathcal{A}$ and then acting greedily according to its estimation $\hat{H}(s, a)$. Results of the experiments led in [23] show that for the same task, a TAMER agent was learning faster than a traditional RL agent. This can be easily explained by the fact that the signal given to the TAMER agent is much richer than the traditional Reinforcement Learning reward signal.

However, human signals may be flawed since the human, although good at solving tasks are not perfect and may under- or over-estimate the consequences of each action. Extensions of the TAMER algorithm have therefore been introduced in order to be able to combine the manual TAMER feedbacks with Reinforcement Learning in a TAMER+RL framework and take the best of both worlds. In [24], several ways of mixing both reward signals and TAMER signals were investigated. Of all the investigated methods, it is shown experimentally that the most effective one is to directly act on policy, without biasing the Q-function approximation, and at each time step t, to take in state s the action a given by $a = \text{argmax}_{a'}[Q(s, a') + \alpha H(s, a')]$, where α is a parameter of the algorithm emphasizing the weights of the advice.

In the following section, we extend this work by providing a way of integrating TAMER feedback via policy biasing in a batch fashion.

3.3 The TAMER-LSPI algorithm

A Batch TAMER+RL procedure may be decomposed into the four steps presented in figure 2. First, a batch of data $\mathcal{D} = [(s_i, a_i, s'_i, r_i)]_{i=1}^{N}$ is collected. In a second step, this batch is analyzed and annotated by the human expert. Annotations take the form of an additional reward function, leading to a batch $\mathcal{D}' = [(s_i, a_i, s'_i, r_i, H_i)]_{i=1}^N$, a positive reward is provided to safe and efficient behaviors, while a negative reward is associated with bad ones. A Batch TAMER+RL algorithm is then applied on the batch $\mathcal{D}'$ to get the most efficient policy out of this set. It is important to emphasize that such an algorithm must be data-efficient with respect to the human annotations. Human guidance is indeed very costly and the annotations provided by the expert are thus likely to be sparse. Finally, once that a policy has been outputed by the Batch TAMER+RL algorithm, the agent is released to the real world where its policy remains unchanged.

Providing the expert advice in a batch setting is particularly interesting from a practical point of view. It is indeed easier to determine a posteriori, *i.e.* when the whole trajectory is given, which were the action which led to the success (or failure) of this trajectory. Conversely, when the feedback is given in an on-line fashion, it is not always obvious to consider the long-term consequences of the action taken by the machine. Furthermore, in a batch fashion, it is also easier to observe good tendencies from what has been observed and thus derive more a more general reward function based on heuristics emphasizing those tendencies.

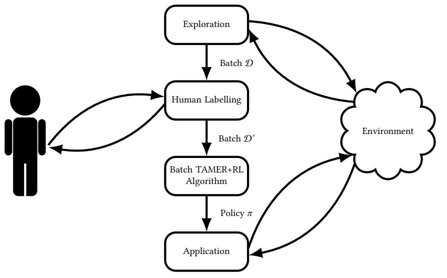

Figure 2: The four steps of a Batch TAMER+RL procedure: 1) Data Gathering with exploration 2) Annotations of the data by the human expert 3) Application of a Batch Reinforcement Learning Algorithm 4) Application of the policy in the real world.

As [24] show, the more an algorithm affects the Q-function with additional reward (as is the case for Reward Shaping), the worse it does. Conversely, the more it affects the policy, the better it does. We therefore designed our Batch TAMER+RL algorithm in order to bias the action selection mechanism of the agent with the additional reward, while learning the accurate values of its policy. As an algorithm working with two independent steps of Policy Evaluation/Policy Improvement, LSPI is the candidate of choice for such framework. Indeed, modifications in the Policy Improvement step do not affect the values learned during the Policy Evaluation.

In order to deal with the sparsity of the advice provided by the human advice and to be able to generalize those advice to unseen states, H is also approximated with a linear regression: $\hat{H}(s,a) = \langle w, \phi(s,a) \rangle$, . The weight vector w is computed by minimizing the loss: $\sum_{i=1}^N (H_i - \phi(s_i, a_i)w) + \lambda ||w||_2$, where λ is a ℓ_2 regularization parameter, via Ridge regression. In the same way that Batch Reinforcement Learning provides a major stability improvement over online methods, regressing the H-function over the whole set of data in a synchronous fashion also leads to more stable estimates of the H-values.

As recalled in theorem 2.1, the key of learning a good policy with LSPI is to provide during the Policy Improvement phase each epoch of the algorithm, a policy better than the previous one. In our case, following [24] and assuming that human advice is *good*, we know that, at each time step t, following the action a greedy with respect to $Q^{\pi_{t-1}}(s,a') + \alpha H(s,a')$ will likely lead to a global policy improvement, bigger than the one provided while being only greedy with respect to the learned value function. $H(s,a)$ making up for the approximation errors obtained while estimating $Q^{\pi}(s,a)$.

All of those modifications lead to Algorithm 3:

```
Input  : An initial policy $\pi_0$,
         A batch $\mathcal{D} = [(s_i, a_i, s'_i, r_i, H_i)]_{i=1}^N$
         Parameter $\alpha$
Output : The optimal policy $\pi^*$
Regress the $H$-function by minimizing the following loss,
 l2-regularised with the $\lambda$ parameter
 $\sum_{i=1}^N (H_i - \phi(s_i, a_i)w) + \lambda ||w||_2$
repeat
    1. Estimates $\hat{Q}^{\pi_n} = \langle \theta_n, \Phi \rangle$ where $\theta_n$ is the solution of the
       equation $(A + \beta I)\theta_n = b$ where where $A$ and $b$ are
       computed from the samples:
       $A = \sum_{i=1}^N \Phi(s_i, a_i)(\Phi(s_i, a_i) - \gamma \Phi(s'_i, \pi_n(s'_i)))^T$,
       $b = \sum_{i=1}^N \Phi(s_i, a_i) r_i$ , $\beta$ is a regularizing constant and $I$
       the identity matrix
    2.Set $\forall s \in \mathcal{S}$, $\pi_{n+1}(s) = \text{argmax}_{a \in \mathcal{A}} \hat{Q}^{\pi_n}(s,a) + \alpha \hat{H}(s,a)$
       with $\hat{H}(s,a) = w^T \Phi(s,a)$
until $\pi_{n+1} = \pi_n$;
```

Algorithm 3: TAMER-LSPI algorithm

By decreasing the parameter α at each epoch, the impact of the human bias also decreases and asymptotically, the Reinforcement Learning agent learns the optimal policy.

It is important to notice that the additional reward is not necessarily a label given by the human expert at each sample. It can encode every kind of shaping rewards without taking care of deriving it from a potential function. For instance, it may correspond to different scores encoding user feedbacks [11], an estimate of the Q-function computed on a simulated user [46]... In section 4, a set of rules encoded into a reward form are used as the TAMER shaping function.

4 EXPERIMENTAL SETTING

4.1 Task

Experiments are led on a restaurant booking dialogue management task. In this task, a human, called *user*, needs to use its dialogue system in order to look for a restaurant suiting his/her personal tastes. In our setting, the human is modeled by a user simulator described in section 4.3. In this work, we address the problem of learning the strategy of the dialogue system in order to respond to the needs of its user in the most effective way.

4.2 The DSTC2 dataset

To make the experiment as realistic as possible, we trained all of our models on the Dialogue State Tracking Challenge 2 (DSTC2) dataset [17]. This dataset consists of 3000 dialogues extracted from a Restaurant Booking system. One key feature of this dataset is that all utterances are annotated as dialogue acts. Furthermore, at each dialogue turns, user goals are explicitly given. In this dataset, the aim of the Dialogue Manager is to help the user to find a restaurant matching his/her constraints and then providing him/her all informations s/he might request, such as address, telephone number... Figure 3 illustrates the kind of dialogues which are found in the DSTC2.

In this dataset, the Dialogue Manager is replaced by a Wizard-of-Oz, allowing thus the gathering of expert 'system' data and providing a user behavior as consistent as possible.

4.3 User Simulator

We constructed the user simulator according to the methodology provided by [9]. Within this framework, the user is modeled with an LSTM which takes as input the entire history of dialogue contexts and outputs the most probable user act.

LSTM [19] are recurrent neural networks able to deal with long-term temporal dependencies. The architecture of the LSTM cell is depicted on figure 4. The LSTM cell consists of different blocks, called cell state, output and gates. Each block has a precise function. The cell state C_t is the main component of the LSTM. It is supposed to represent the past in the most relevant way possible. Only linear modifications are done to the cell state. Those modifications may be a removal of irrelevant past information, which is controlled by the forget gate f_t. But relevant present information may also be added to the cell state, the input gate i_t is in charge of this operation. The output h_t is a filtered form of the cell state, which is transformed in order to fulfill the objectives of the designer.

The LSTM cell is implemented using the following equations:

$$
\begin{aligned}
i_t &= \sigma(W_i c_t + U_i h_{t-1}) \\
f_t &= \sigma(W_f c_t + U_f h_{t-1}) \\
C_t &= i_t * \tanh(W_c c_t + U_c h_{t-1}) + f_t * C_{t-1} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1}) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

with i_t the input gate, σ the sigmoid function, f_t the forget gate, o_t the output gate, C_t the cell gate and h_t the hidden state.

To construct the simulator, in the beginning of each dialogue, a goal $G = (C, R)$ is randomly sampled. This goal contains the constraints C and the requests R of the user. Constraints are the values that have to be given to the dialogue system, while requests are the values the user has to receive from the system. The user has at most three constraints to give to the system and may request three informations from it.

A dialogue context c_t at time t is defined by the following components: the last dialogue act a_t given by the dialogue manager, an inconsistency vector i_t tracking the inconsistencies between the informations given by the machine and the user goal, a constraint vector c_t and a request vector r_t tracking the informations of the user goal given to the machine.

The machine act vector a_t is a vector of n_{ma} components, with n_{ma} being the number of machine acts. This vector has ones for each dialogue act outputed by the machine in the previous turn and zeros otherwise.

The inconsistency vector i_t is given by two vectors of size n_c, where n_c is the number of possible user constraints. When the system proposes a restaurant to the user, the first vector verifies that all of the user constraints have been respected. The second vector tracks the inconsistencies of the dialogue manager when it mentions a slot in every other situation (*e.g.* in a confirmation). Both of these vectors have ones for every violated constraints and zeros everywhere else. These vectors are reseted after each turn.

Constraint and request vectors c_t and r_t track what are the values of the user goal which have already been given to the system and correctly understood by it. In the beginning of each dialogue, those vectors have zeros for every constraints and requests in the user goal and ones everywhere else. If a constraint is set to zero in the inconsistency vector, it is reset to one in the constraint vector. The request vector is reset after each proposition of the machine.

Those context vectors are used as inputs in order to feed an LSTM neural network with 64 units.

We add to this LSTM a softmax layer in order to output probabilities over the possible actions of the user. This neural network is then trained on the DSTC2 dataset. Actions of the user are finally drawn according to the probabilities given by the softmax layer.

Finally, a set of rules overcoming the action chosen by the LSTM are handwritten in order to ensure a coherent behavior. First, since the most important advice that we will provide to the system are mostly rules concerning when and how to ask for a confirmation, we imposed to the user to repeat its last sentence when the system asks for it, and to acknowledge or to deny if the system asks for a confirmation. Second, to ensure that the dialogue will end, with a probability $p = 0.02$ or if the user has given/received all the informations that s/he needed, s/he ends the dialogue.

4.4 Dialogue system state space and ASR model

In order to apply reinforcement learning methods to train the dialogue system, a markovian state space has to be designed. In the following experiments, a neural-based Reinforcement Learning method [2] has been chosen to learn one. The main idea behind our approach is to consider an LSTM similar to the user simulator but learning to output the dialogue acts of the system in the DSTC2 dataset. Since the cell state of this LSTM contains enough information to predict the future, it respects the Markov property and may be used as the state of the dialogue manager.

More precisely, we first implemented an LSTM to output the most probable action of the system given the entire history of dialogue contexts. Those contexts consist of three vectors, the user act vector a'_t, the constraint vector c'_t and the request vector r'_t. The user act vector a'_t has n_{ua} components, where n_{ua} is the number of possible system dialogue acts. This vector has ones for each dialogue act given by the user in the previous turn and zeros everywhere else.

Constraint and request vectors c'_t and r'_t have ones for each of the constraints given by the user to the system and zeros everywhere

System	Hello, welcome to the Cambridge restaurant system. You can ask for restaurants by area, price range or food type. How may I help you?	`welcomemsg`
User	Moderately priced swedish food	`inform`[Swedish]
System	Sorry there is no swedish restaurant in the moderate price range	`canthelp`
User	How about asian oriental?	`inform`[Asian oriental]
System	Yippee noodle bar is a great restaurant serving asian oriental food and it is in the moderate price range	`offer`[Yippee noodle bar]
User	Could i have the address and phone number?	`request`[phone,address]
System	Sure , yippee noodle bar is on 40428 King Street City Centre	`offer`[address]
System	The phone number of yippee noodle bar is 01223 518111	`offer`[phone]
User	Thank you good bye	`bye`

Figure 3: Dialogue example - The first column corresponds the locutor, the second is the transcription of the utterance and the third is the dialogue acts outputed by the Spoken Language Understanding Module

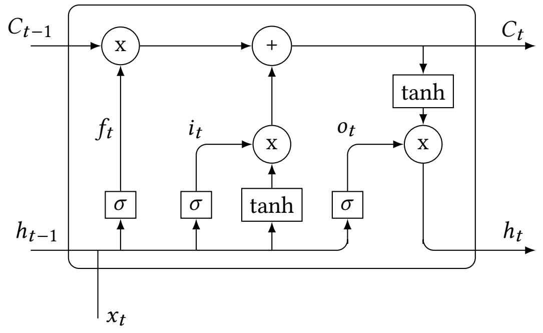

Figure 4: LSTM cell

else. In the case of an ASR error, a random slot is set to one in the constraint vector.

Similarly to the user simulator, a softmax layer is added to the LSTM, outputs of this layer corresponding to probabilities of the different actions of the system.

Experimentally, we found that such a representation was prone to Internal Covariate Shift [20], which led to bad Q-function representations. This issue is overcome by stabilizing the hidden state dynamics with Layer Normalization [1].

It is also assumed in this work that the ASR module of the dialogue system is not perfect and that misunderstandings may happen. When the user gives one of his/her constraints to the system, with probability $p = 0.3$, the system understands something else. An ASR score is then computed following the methodology of [21]. A random number x is first drawn according to a normal distribution centered in +1 in the case of correct understanding and centered in -1 in the case of a misunderstanding. The final score is then computed by passing x through a sigmoid function : $score = \frac{1}{1-e^x}$.

We assumed that all other user acts are always correctly understood by the system. In that case, the ASR score is 1.

Since this information about the ASR score is not present in the LSTM, we constructed the state space of the dialogue system by concatenating the cell state vector c'_t of this LSTM with the ASR score and the time step t.

4.5 Dialogue system action space

The action space of the system contains all the different machine dialogue acts present in the corpus. To keep things simple, when the system suggests a restaurant, we assumed that there is always a restaurant satisfying all of the constraints understood by the system. For the same reason, if the system answers a request of the user, one assumes that it can always provide an answer to this request.

4.6 Reward function

In the experiments, we used a traditional reward scheme for DSTC2. If the dialogue finishes successfully (*i.e.* a restaurant satisfying all of the constraints of the user and if all of his/her requests have been answered), a reward of +1 is given to the system. However, if the user (or the system) hangs up before realizing all of his/her goals, no reward is given to the system.

To ensure that the system will try to make the dialogue as short as possible, the discount factor γ is set to 0.9.

4.7 TAMER rewards

To provide good human advice, a simple effective policy is handcrafted. At each time step t, taking in state s the action a prescribed by the rules leads to a TAMER reward $H(s, a)$ of +1, while taking another action leads to no additional reward.

The handwritten rules take only into account the last user dialogue act and the ASR score. At the beginning of the dialogue, the system welcomes the user. If the ASR score is smaller than 0.3, it asks the user to repeat what has been said; if it is between 0.3 and 0.7, it asks for a confirmation; and if it is greater than 0.7, it requests a slot from the user. If the user asks to repeat, the system repeats. when s/he requests something about a slot, the system Inform her/him about it. Finally, otherwise, the system requests a slot from the user.

Finally, in order to avoid an early end induced by the system, we added a -1 TAMER reward if the system ends the dialogue.

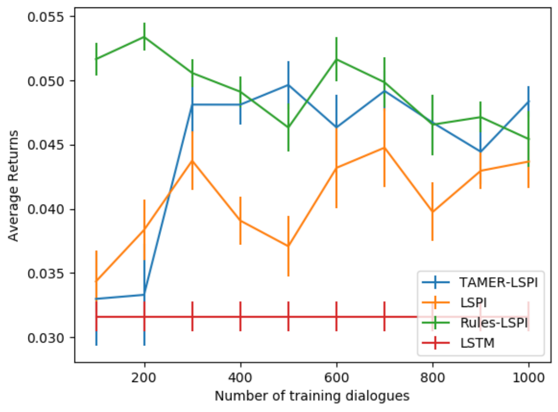

Figure 5: Average Discounted returns

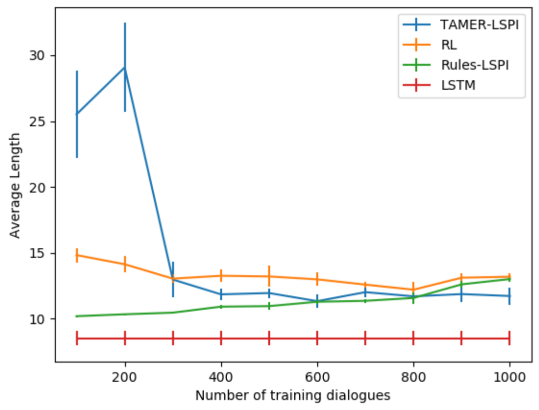

Figure 6: Dialogue Lengths

5 RESULTS

Performances of the TAMER-LSPI algorithm are compared with LSPI as a pure Batch Reinforcement Learning algorithm, a mixture of rules with LSPI, and the generative policy provided by the LSTM simulating the system. In all the experiments, the LSPI regularization parameter β is set to 2 and the TAMER regularization parameter λ is set to 1. In every case, a batch of 200 dialogues is first collected by following the handwritten strategy defined in the previous section. Each learning algorithm is then train on those data. The obtained policy is then used to gather more data, 50 dialogues with this policy and 50 dialogues following an ϵ-greedy exploration strategy, with ϵ set to 0.1. This process of policy learning/data gathering is redone for 20 epochs, until 2000 dialogues have been gathered.

At each epoch k, the parameter α of the TAMER-LSPI algorithm is set to $1/k$. The rule-based/RL mixture is done by applying the RL policy and picking the action prescribed by the rules with a probability $1 - 0.1 * (k - 1)$: in the beginning, the policy is fully rule-based and in the end, RL based. In order to model the incompleteness of the rules provided in a real world dialogue scenario, we introduced stochasticity in them. Thus, when the system has to follow the rules, with a probability $p = 0.25$, it takes a random action not prescribed by the rules.

Figure 5 shows the performances of the four policies at each epoch in terms of discounted returns, while figure 6 shows the average dialogue lengths. Results are computed by averaging the performances of 20 runs, with each one trained from scratch.

On both figures, one sees that with a small number of samples, the performance of the rule-based/RL policy outperforms the policies provided by the LSTM, LSPI and TAMER-LSPI. This is easily explained by the fact that the handcrafted policy is almost optimal and that no learning algorithm may learn such a good policy with as few samples.

On the other hand, while TAMER+RL is worse than the Rule-Based policy on the first epochs, it always performs better than pure Reinforcement Learning. Not surprisingly, in the beginning, the extra information provided by advice allows TAMER-LSPI to outperform RL, whose policies suffer from a lack of samples. Interestingly, it also seems to perform better than RL asymptotically. This phenomenon could be explained by the fact that TAMER-LSPI was able to explore more interesting regions of the state space during the data gathering phase.

Furthermore, figure 6 shows that in the beginning, the average dialogue length of TAMER-LSPI is longer than the others, but this dialogue length drastically also diminishes around 300 training dialogues. This means that even systems learned in the beginning not to use the end action thanks to advice, it then learns to go beyond those advice and takes them into account only when they are beneficial, which shows robustness against bad human bias (*i.e.* non-pertinent advice).

6 CONCLUSION

In this paper, we introduced a novel method to incorporate human advice during the training phase of statistical dialogue systems in order to provide safe behavior. It relies on the TAMER+RL framework. Experimental results show that the use of such advice signals may beneficially affect the performance of Reinforcement Learning algorithms for all stages of the training phase, in the beginning, where the lack of training samples often leads to bad and unsafe RL policies, but also asymptotically. Results also show that TAMER+RL is robust against bad human bias. Our method finally presents the advantage of being completely statistical. It is therefore compatible with more complex Deep Learning models which should be able to deal with real world situations where no efficient hand-crafted policy may be designed and where advice may be sparse (*i.e.* only a few rules accompanied with labeled samples). A natural future direction of this work would be to confront our framework with such situations.

ACKNOWLEDGEMENT

This work has received funding from the European Commission H2020 framework program under the research Grant number 687831 (BabyRobot)

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[2] Bram Bakker. 2002. Reinforcement learning with long short-term memory. In *Proc.of NIPS*.

[3] Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefevre, and Olivier Pietquin. 2011. User simulation in dialogue systems using inverse reinforcement learning. In *Interspeech 2011*. 1025–1028.

[4] Lu Chen, Runzhe Yang, Cheng Chang, Zihao Ye, Xiang Zhou, and Kai Yu. 2017. On-line Dialogue Policy Learning with Companion Teaching. *Proc. of EACLK* (2017).

[5] Sonia Chernova and Manuela Veloso. 2009. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research* 34, 1 (2009), 1.

[6] Jeffery Allen Clouse. 1996. *On integrating apprentice learning and reinforcement learning*. Technical Report. Amherst, MA, USA.

[7] Jeffery A Clouse and Paul E Utgoff. 1992. A teaching method for reinforcement learning. In *Proc. of ICML*.

[8] Lucie Daubigney, Matthieu Geist, and Olivier Pietquin. 2012. Off-policy learning in large-scale POMDP-based dialogue systems. In *Proc. of ICASSP*.

[9] Layla El Asri, Jing He, and Kaheer Suleman. 2016. A sequence-to-sequence model for user simulation in spoken dialogue systems. In *Proc. of Interspeech*.

[10] Layla El Asri, Romain Laroche, and Olivier Pietquin. 2013. Reward shaping for statistical optimisation of dialogue management. In *Proc. of SLSP*.

[11] Layla El Asri, Bilal Piot, Matthieu Geist, Romain Laroche, and Olivier Pietquin. 2016. Score-based inverse reinforcement learning. In *Proc. of AAMAS*.

[12] Mehdi Fatemi, Layla El Asri, Hannes Schulz, Jing He, and Kaheer Suleman. 2016. Policy networks with two-stage training for dialogue systems. *Proc. of SigDial*.

[13] Milica Gašić, Filip Jurčíček, Simon Keizer, François Mairesse, Blaise Thomson, Kai Yu, and Steve Young. 2010. Gaussian processes for fast policy optimisation of POMDP-based dialogue managers. In *Proc. of SigDial*.

[14] Geoffrey J Gordon. 1995. Stable function approximation in dynamic programming. In *Proc. of ICML*.

[15] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles Isbell, and Andrea L Thomaz. 2013. Policy shaping: Integrating human feedback with reinforcement learning. In *Proc. of NIPS*.

[16] Anna Harutyunyan, Sam Devlin, Peter Vrancx, and Ann Nowé. 2015. Expressing Arbitrary Reward Functions as Potential-Based Advice.. In *Proc. of AAAI*.

[17] Matthew Henderson, Blaise Thomson, and Jason Williams. 2014. The second dialog state tracking challenge. In *Proc. of SigDial*.

[18] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. 2017. Learning from Demonstrations for Real World Reinforcement Learning. *arXiv preprint arXiv:1704.03732* (2017).

[19] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[20] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. of ICML*.

[21] Hatim Khouzaimi, Romain Laroche, and Fabrice Lefevre. 2015. Optimising turn-taking strategies with reinforcement learning. In *Proc. of SigDial*.

[22] Beomjoon Kim, Amir massoud Farahmand, Joelle Pineau, and Doina Precup. 2013. Learning from limited demonstrations. In *Proc. of NIPS*.

[23] W Bradley Knox and Peter Stone. 2009. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proc. of ICKC*.

[24] W Bradley Knox and Peter Stone. 2010. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of AAMAS*.

[25] J Zico Kolter and Andrew Y Ng. 2009. Regularization and feature selection in least-squares temporal difference learning. In *Proc. of ICML*.

[26] Michail G Lagoudakis and Ronald Parr. 2003. Least-squares policy iteration. *Journal of machine learning research* 4, Dec (2003), 1107–1149.

[27] Sascha Lange, Thomas Gabel, and Martin Riedmiller. 2012. Batch reinforcement learning. In *Reinforcement learning*. Springer, 45–73.

[28] Romain Laroche, Ghislain Putois, and Philippe Bretier. 2010. Optimising a handcrafted dialogue system design.. In *Proc. of Interspeech*.

[29] Romain Laroche and Paul Trichelair. 2017. Safe Policy Improvement with Baseline Bootstrapping. *arXiv preprint arXiv:1712.06924* (2017).

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[31] Esther Levin and Roberto Pieraccini. 1997. A stochastic model of computer-human interaction for learning dialogue strategies.. In *Proc. of Eurospeech*.

[32] Lihong Li, Jason D Williams, and Suhrid Balakrishnan. 2009. Reinforcement learning for dialog management using least-squares Policy iteration and fast feature selection.. In *Proc. of Interspeech*.

[33] Ryan Thomas Lowe, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. 2017. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue & Discourse* 8, 1 (2017), 31–65.

[34] Richard Maclin and Jude W Shavlik. 1996. Creating advice-taking reinforcement learners. *Machine Learning* 22, 1-3 (1996), 251–281.

[35] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. of ICML*.

[36] Olivier Pietquin, Matthieu Geist, Senthilkumar Chandramohan, and Hervé Frezza-Buet. 2011. Sample-efficient batch reinforcement learning for dialogue management optimization. *ACM Transactions on Speech and Language Processing (TSLP)* 7, 3 (2011), 7.

[37] Olivier Pietquin and Steve Renals. 2002. ASR system modeling for automatic evaluation and optimization of dialogue systems. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, Vol. 1. IEEE, I–45.

[38] Bilal Piot, Matthieu Geist, and Olivier Pietquin. 2014. Boosted Bellman residual minimization handling expert demonstrations. In *Proc. of ECML*.

[39] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[40] Ghislain Putois, Romain Laroche, and Philippe Bretier. 2010. Online reinforcement learning for spoken dialogue systems: The story of a commercial deployment success. In *Proc. of SIGDIAL*.

[41] Jette Randlov and Preben Alstrom. 1998. Learning to drive a bicycle using reinforcement learning and shaping. In *Proc. of ICML*.

[42] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proc. of AISTATS*.

[43] Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The knowledge engineering review* 21, 2 (2006), 97–126.

[44] Satinder P Singh, Michael J Kearns, Diane J Litman, and Marilyn A Walker. 2000. Reinforcement learning for spoken dialogue systems. In *Proc. of NIPS*.

[45] Florian Strub, Harm de Vries, Jeremie Mary, Bilal Piot, Aaron Courville, and Olivier Pietquin. 2017. End-to-end optimization of goal-driven and visually grounded dialogue systems. *Proc. of IJCAI*.

[46] Pei-Hao Su, David Vandyke, Milica Gasic, Nikola Mrksic, Tsung-Hsien Wen, and Steve Young. 2015. Reward shaping with recurrent neural networks for speeding up on-line policy learning in spoken dialogue systems. *Proc. of SigDial* (2015).

[47] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[48] Blaise Thomson, Milica Gasic, Matthew Henderson, Pirros Tsiakoulis, and Steve Young. 2012. N-best error simulation for training spoken dialogue systems. In *Spoken Language Technology Workshop (SLT), 2012 IEEE*. 37–42.

[49] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proc. of SigDial*.

[50] Eric Wiewiora, Garrison Cottrell, and Charles Elkan. 2003. Principled methods for advising reinforcement learning agents. In *Proc. of ICML*.

[51] Jason D Williams. 2008. The best of both worlds: unifying conventional dialog systems and POMDPs.. In *Proc. of Interspeech*.

[52] Jason D Williams and Steve Young. 2003. Using Wizard-of-Oz simulations to bootstrap Reinforcement-Learning based dialog management systems. In *Proc. of SIGDIAL*.