Statistical Spoken Dialogue Systems

Riashat Islam
Department of Engineering
University of Cambridge
Trumpington Street, Cambridge, CB2 1PZ, England
ri258@cam.ac.uk

I. Introduction

Spoken Dialogue Systems (SDS) are used to allow users to be able to interact with systems via speech. In each section, we present our experimental results followed by a discussion and summary of results. We then evaluate and compare all the experimental results, and include a future direction of work. Code snippets are included in appendix.

II. Background

We first give a brief introduction to the two key ideas to a Partially Observable Markov Decision Process (POMDP) based dialogue system: belief state tracking and reinforcement learning [1].
Belief state tracking in SDS is used to update the posterior probability of the belief state after each user input using Bayesian inference. The system maintains a belief distribution over all states such as to capture all possible dialogue paths. The states are encoded into three distinct types of information: user's goal, most recent utterance and the dialogue history.

In a spoken dialogue system, a dialogue manager is used such as to select actions based on observations of events (what the user says) and inferred beliefs using belief state tracking. In reinforcement learning based SDS, the goal is to learn a policy such as to optimize the action selection process (what the system says back to user) such as to maximize a reward function. Rewards are associated with each state action pairs and it gives an objective measure of the performance of the system both off-line using dialogue corpora and on-line through interaction with real or simulated users. Decision-making and policy optimisation takes place in the summary space, which is a compressed summary space in which states and actions are simplified. Summary spaces are considered as a subspace of the master space, and so a dictionary is maintained consisting of state action pairs. Therefore, the policy can be considered as a function of the summary belief state and actions, instead of the original belief state and actions.

III. Dialogue State Tracking

A. Method

We consider the evaluation of two state trackers: baseline and focus tracker on the DSTC dataset which is used as part of the Dialogue State Tracking Challenge. Baseline trackers gives a single hypothesis for each slot whose value is the top scoring suggestion so far in the dialogue. Focus trackers, in comparison, include a model of changing goal constraints. In focus tracker, beliefs are updated for the goal constraint $s = v$ at turn t, $P(s = v)$ using the rule $P(s = v)_t = q_t P(s = v)_{t-1} + SLU(s = v)_t$ where SLU is the evidence given $s = v$ in turn t.

B. Experimental Results

The output of both baseline and focus tracker is a distribution over dialog states in each turn. The dialog state is composed of three components: the goal constraint for each slot, the requested slots and the method. **Accuracy - acc** values are used to measure the fraction of turns when the top hypothesis was correct. **Average Probability - avgp** then measures the mean score of the correct hypothesis and evaluates the quality of the score given the correct hypothesis. **L2** measures the square norm between the distribution and the correct label to measure the quality of the whole distribution. Finally **ROC** values measures the discrimination of the scores for the highest ranked hypothesis. There are 815 total metrics that are calculated per tracker in evaluation [2], [3].

For each of our trackers, we consider comparing only three evaluation metrics: accuracy, L2 norm and the ROC values. We also compare ROC metrics for each tracker at similar accuracies, and show that accuracy and ROC metrics are maximized while L2 values must be minimized.

CODE: The code for the implementation of the focus dialogue state tracker in given in appendix section.

Baseline Tracker			
	Joint Goals	Requested	Method
Accuracy	0.568	0.914	0.682
L2	0.834	0.112	0.576
ROC	0.00	0.606	0.002
Focus Tracker			
	Joint Goals	Requested	Method
Accuracy	0.732	0.881	0.675
L2	0.421	0.189	0.576
ROC	0.00	0.003	0.002

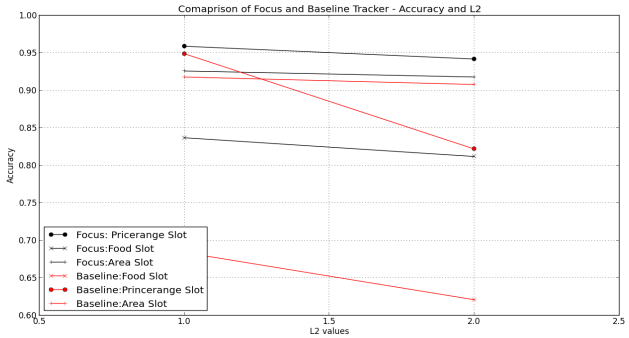

Fig. 1. Accuracy and L2 scores on Area, Food and Pricerange slot for Focus and Baseline Tracker

C. *Discussion of Results*

The results show that **accuracy** measures for focus tracker is higher 0.732 for Joint Goals than baseline tracker 0.568. This is an important measure since accuracy shows the number of turns when the top hypothesis is correct. Our results therefore suggest that by accumulating evidence of how the state changes throughout the dialogue, instead of only using the current state and ignoring past states as in baseline tracker, we can achieve higher correctness of hypothesis. The results of the L2 scores also shows better minimization for the Focus Tracker compared to Baseline for the Joint Goals, as lower L2 scores are better.

Figure 1 investigates the correlation between accuracy and L2. Figure shows the metrics for each focus and baseline tracker on joint goal constraints, and we can conclude that in general a lower L2 score correlates with a higher accuracy. By maintaining a distribution of belief states and tracking changes in the states, focus tracker can achieve better results compared to baseline.

D. *Future Work*

Word-based dialogue state tracking can be performed using a recurrent neural network based approach, for discriminative learning over dialogue sequences. Recent work focussed on the tracking of goal states using RNN, and considered an online unsupervised adaptation to new domains using word-based RNNs. Such a method is shown to outperform in the third DSTC challenge [4]. Recent work also considered a deep neural network based approach for tracking the dialogue state, where rich feature representations of the dialogue can improve performance [5]. Word-based state tracking directly on the output of a speech recogniser instead of considering semantic representation can perform significantly better.

IV. Monte Carlo Control

A. *Method*

Monte Carlo control learns by estimating the average return from complete episodes to update the action-value Q function. It therefore learns without bootstrapping. The policy improvement step considers an ϵ-greedy method with respect to Q function to get improved estimates of the Q function from which policy can be derived. The Q function is estimated iteratively whilst interacting with the simulated user. Monte-Carlo policy evaluation uses empirical mean return instead of the expected return. Since we consider complete episodes in MC methods, updates are only performed at the end of each dialogue and therefore the total discounted return (whole-dialogue rewards) are used to directly improve the Q function [6]. The total discounted return can be given as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} ... \gamma^{T-1} R_T \tag{1}$$

$$q(i(t), \hat{a}) = q(i(t), \hat{a}) + G_t \tag{2}$$

$$N(i(t), \hat{a}) = N(i(t), \hat{a}) + 1 \tag{3}$$

$$Q(\hat{b}_i, \hat{a}) = \frac{1}{n(i, \hat{a})} q(i, \hat{a}) \tag{4}$$

Code: Monte-Carlo Control is given in appendix section.

B. *Experimental Results*

In Monte Carlo based control for learning the action-value function for policy updates, a dictionary is maintained which contains the pairs of beliefs and actions. At first, using a simulated user we generate the state action trajectories and rewards and train our policy such as to learn the mappings of actions to take at every state. Therefore the Q function can be estimated at any point.

In our experiments, we consider varying the size of the threshold to determine whether the model should update the Q of the grid point. However, if the threshold value is too small, then too many grid points will be used which can lead to poor performance due to over-fitting. Conversely, if the threshold value is too high, then the current (b,a) pair will be added to the dictionary and the Q function will be initialised with sampled total return. If the threshold value is too high, it means that less of the Q(b,a) will be updated (since the distance between the (b,a) pair and the nearest grid point is larger than the threshold). Note that the Monte Carlo control algorithm already has high variance in the Q function estimates since it considers the entire trajectory for estimating mean return rather than going using step by step on-policy like Q-learning.

Figure 2 shows the test policy performance for different threshold values. Figure 3 finally shows the test policy performance (with uncertainty estimates) based on the optimal threshold value that achieves best performance having trained a policy based on simulated user. Each iteration considers 100 episodes. We showed the averaged test results of 100 dialogues after every 100 trained dialogues, to show how the learnt policy improves based after training every 100 dialogues.

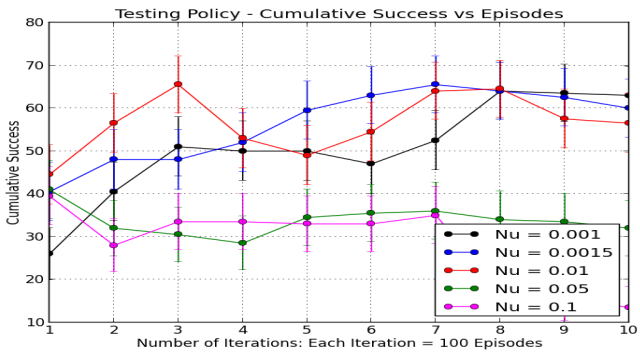

Fig. 2. Policy Testing with Different Threshold Values

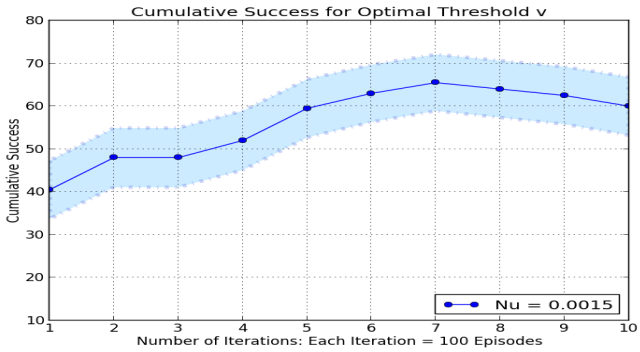

Fig. 3. Cumulative Success for Monte Carlo Policy with Optimal Threshold parameter μ

C. *Discussion of Results*

Our results show how the cumulative reward evaluated from learning the policy by updating the Q function varies with the number of episodes used during training and testing. During training, the results show that with a small threshold (0.001 or 0.0015), the cumulative reward that can be achieved is higher at both training and testing. This is because with small threshold values, more grid points are added to the dictionary and the action-value function is evaluated more frequently. This leads to better estimates of the policy, and hence better learning of action mapping at every state.

Since Monte Carlo methods learn from complete episodes only, the results also suggest that as we train policies with more dialogues, the test performance of the learnt policy improves. By using a small threshold value, we have higher number of policy evaluations (estimation of Q) where the trajectory data for training was also obtained by a greedy policy improvement method ensuring exploration of the environment. Large number of steps in a given episode is also better for estimating the Q function since we are also maintaining an eligiblity tree (N) of states and actions.

V. Gaussian Process-SARSA

A. *Method*

In this section, we further analyse the use of Gaussian Processes for policy modelling. We examine the different formulations for a GP-based policy to minimise the variability in the learning process [7].

SARSA algorithm is a form of temporal difference learning, and we evaluate the advantages of TD learning compared to Monte-Carlo estimates. Our results in this section shows that using a GP-SARSA algorithm, we can increase the policy learning rate, and overcome the issue of using large amount of training data to train a dialogue policy and having to choose a suitable policy parameterisation by considering non-parametric approaches to policy modelling using Gaussian Processes [8], [9].

Using a Bayesian approach for policy optimisation in both model-free and model-based reinforcement learning framework, we consider incorporating prior knowledge about the task such that it can improve the policy learning rate. Our work involves analysing the significance of the kernel function in the GP-based nonparametric policy optimisation. Furthermore, considering the Bayesian approach for policy learning, during exploration, actions are chosen according to the variance of the GP estimate for the Q function, whereas during exploitation the actions are chosen according to the mean.

B. *Advantages of SARSA over Monte-Carlo*

Compared to MCC, SARSA can learn from incomplete episodes by bootstrapping, before knowing the final outcome. In other words, SARSA can learn online after every step, compared to MC methods that need to wait until end of the episode [6]

From a dialogue systems point of view, this further means that SARSA can improve the Q function estimates after each dialogue step. This means that while within an episode (dialogue) SARSA can improve Q function after each (b,a) pair encountered. In other words, the dialogue manager can improve the policy after each user interaction within an episode, unlike MC methods that requires a complete dialogue interaction with the simulated user. Using SARSA, this online form of policy optimisation is particularly useful in dialogue systems that require real-time interactions with the user. Our experimental results in section V-C further demonstrates this.

Code: GP-SARSA: The code snippet for the GP-SARSA algorithm in given in appendix below.

C. *Experimental Results*

In on-policy SARSA algorithm, the update of the Q function is replaced by the update for the GP Q function posterior. However, since the exact computation of the posterior of the Q function is intractable (the computational complexity of the Q function posterior is $O(t^3)$), we use a sparse approximation and restrict the use of the data points for the Q function approximation. Considering sparse approximation methods for Gaussian Processes, we want to bring down the computational complexity. This is achieved by using a kernel

span sparsification method where a representative set of data points can be achieved as the belief state action space is traversed during on-line interaction with the user.

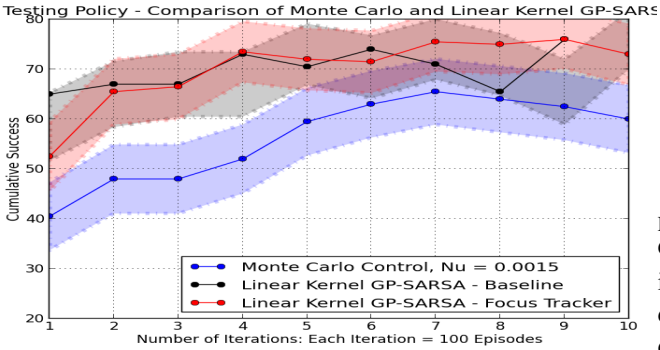

Fig. 4. Comparison between Monte Carlo Control and GP-based SARSA algorithm using Linear Kernel. Comparison with both baseline and focus tracker

Figure 4 shows the comparison of results for GP-SARSA and MC methods. The results show that the on-policy SARSA algorithm (both using baseline and focus tracker) significantly outperforms the MC method in policy testing. Furthermore, our results show that using a focus trakcer for GP-SARSA, better results can be achieved compared to baseline tracker with GP-SARSA.

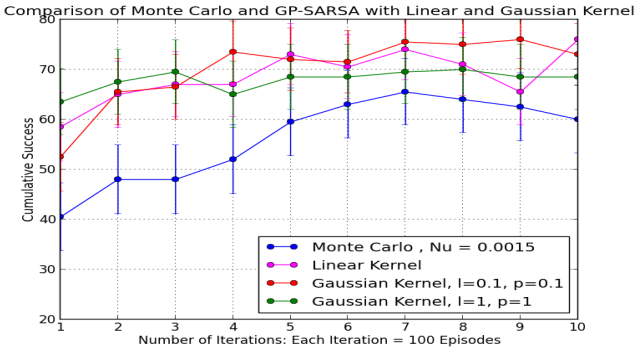

Fig. 5. Comparison of GP-based Linear and Gaussian Kernel with Monte Carlo Control for Baseline Belief Tracker

For the baseline belief state tracker, we then evaluate our results comparing MCC based policy optimisation, with both the Linear Kernel and Gaussian Kernel based GP-SARSA based algorithm, as shown in figure 5.

D. Discussion of Results

Figure 4 shows that on-policy learning with incomplete episodes therefore learns better policy to interact with the user. As discussed earlier in section V-B, on-policy SARSA algorithms achieve better test policy since we obtain better estimates of Q(b,a) for each pair of states and actions. SARSA algorithm also has lower variance compared to MC methods, and the use of incomplete episodes to estimate Q makes

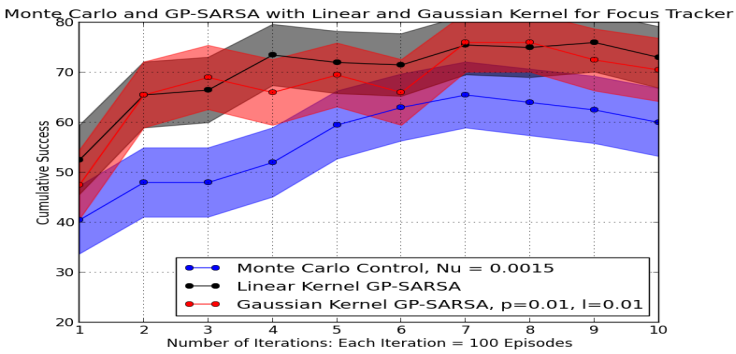

Fig. 6. Comparison for Focus Tracker for Linear and Gaussian Kernel in GP-SARSA and Monte Carlo Control

it avoid bad states in the MDP. Over the course of 1000 dialogue episodes, GP-SARSA outperforms MC methods, due to its online learning property based on each (b,a) pair. Better cumulative reward can be achieved since the linear kernel uses the correlations between the current state-action pair with all other collected state-action pairs. The use of correlation in the on-policy SARSA algorithm is useful since it better helps to find the posterior estimates of the Q function.

In figure 5 we further evaluated the significance of using a Gaussian kernel instead of the linear kernel function. The Gaussian kernel is an exponentially decaying function in the feature space and the intuition comes from a Gaussian distribution. It considers the Euclidean distance between two feature vectors and decreases with the distance between the features. Hence, in finding the correlations between the current state-action pair and all other pairs, it uniformly decays in all the directions away from the current state-action pair.

Figure 5 therefore shows how the fine-tuning of the length scale and variance parameters affects the performance of the test policy in GP-based SARSA algorithm. We see that by careful fine-tuning of l and p parameters, Gaussian kernel based GP-SARSA can achieve a better generalised policy compared to the linear kernel.

Furthermore, note that since SARSA is on-policy, it might reach an optimal reward value (converges faster to optimal Q function), and hence this requires less data for SARSA algorithm compared to MCC. Monte-Carlo, requiring estimated mean over the entire trajectory would therefore require more user data for converging to an optimal Q function. Figure 6 further justifies this, since the cumulative success value (cumulative reward objective function) converges faster for GP-SARSA compared to Monte-Carlo control.

VI. Comparison of Results

In figure 7, we compare our different policy optimisation based approaches with the type of belief state tracking used. Our

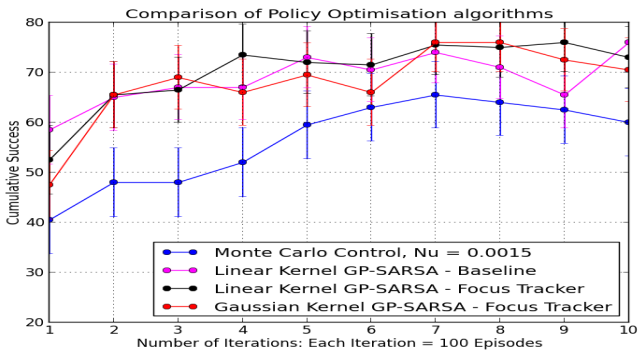

Fig. 7. Comparison of all policy optimisation algorithms

results show that the focus tracker achieves a better belief state tracking for the dialogue agent, and based on this, a better optimal policy can be found over the number of episodes. The linear and Gaussian kernel based GP-SARSA with Focus Tracker outperforms the baseline belief tracker based linear kernel SARSA. Furthermore, we can draw the conclusion that the Gaussian-Process based SARSA algorithm provides better estimates of the Q function compared to Monte-Carlo Control.

VII. Work Extensions and Future Work

Few possible directions of future work and extensions to the work done here:

1) **Policy Gradient approaches:** We can consider policy gradient based approach for policy optimisation, instead of taking SARSA or Q-Learning approach. Specifically, we can extend our work to deterministic policy gradient methods for learning discrete actions to take at every state. However, such methods are not guaranteed to converge to the global optima, and for large state spaces, the convergence of the cumulative reward objective function for a parameterised policy π_θ can be difficult [10], [11], [12].
2) **Neural Network Parameterised Policies:** We can consider the approach of a policy parameterised by neural networks. Taking a policy gradient based approach, we then need to optimise the policy parameters, where taking the gradient of the objective function might be difficult. However, such parameterisation of policies may further allow room for hierarchical reinforcement learning based approaches - where a policy maybe subdivided into multiple policies, in which case, the dialogue manager can learn multiple policies for carrying multiple interactions with the user [13].
3) **Q Function with Neural Network Function Approximation:** Considering a function approximator to the Q function, where we can use a non-linear function approximator such as a neural network, we can perform SARSA and Q-Learning for a DQN based system. This would require also estimating the neural network parameters for learning the Q function based on the TD learning approach. Similar to the atari framework, we can also consider experience replay for DQN based reinforcement learning in a dialogue systems framework [14].
4) **Dialogue State Tracking with Recurrent Neural Networks:** We can consider RNN based approaches for open-domain dialogue systems, where we would need to train a multi-domain RNN dialogue state tracking model. Recent work have shown that using a RNN based belief tracker, the models can capture multiple topics at once. It takes a hierarchical traning procedure to make the model adapt to new domains. Such an approach, combined with a hierarchical RL based policy optimisation framework can further allow the dialogue manager optimise multiple policies at the same time. This is further based on the approach of open-domain dialogue systems.
5) **Hierarchical Reinforcement Learning in Dialogue Systems:** We can consider the HRL framework for optimal dialogue strategies in large state spaces [15], and consider approaches to how RNN based belief tracking combined with a neural network function approximator for Q functions in GP-SARSA can be considered in a hierarchical RL framrwork for learning multiple sub-task policies.

VIII. Conclusion

In this work, we therefore examined the significance of belief state tracking and policy optimisation methods in dialogue systems. We examined the significance of capturing user behaviour by maintaining a belief distribution over the state space, and how this further helps the dialogue manager. Our results showed that using reinforcement learning algorithms, the dialogue manager can better select what to say back to the user. We showed how using a Bayesian approach in policy-optimisation methods, the learned policy can be improved by incorporating prior knowledge using a Gaussian Process based policy optimisation method. Finally, we concluded that by using a focus tracking based belief state tracker, combined with a Gaussian Process based SARSA algorithm for policy optimisation, we can optimise the dilogue policies for a dialogue manager.

IX. Appendix

A. *Code: Dialogue State Tracking*

```
for slot in set(this_u.keys() + hyps["goal labels"]
    if slot not in hyps["goal labels"]:
        hyps["goal labels"][slot] = {}


for value in set(hyps["goal labels"][slot].keys()
+ this_u[slot].keys()):

    if value not in hyps["goal labels"][slot]:
        hyps["goal labels"][slot][value] = 0.0
    hyps["goal labels"][slot][value] =
```

```
    this_u[slot][value] +
    (1    sum(this_u[slot].values()))
    * hyps["goal labels"][slot][value]

# normalise the score of each value in a slo
hyps["goal labels"][slot] =
normalise_dict(hyps["goal labels"][slot])
        #              #

#     2. method     #
method_label = hyps["method label"]

# your code here, modify the following update rule
for method in set(method_label.keys()
+ method_stats.keys()):
    if method != 'none':
        if method not in method_label.keys():
            method_label[method] = 0.0
        method_label[method]
        = method_stats[method] +
        method_stats['none']*
        method_label[method]

        print method_label[method]
# normalise the score
hyps["method label"] =
normalise_dict(method_label)


#     3. requested slots     #
informed_slots = []
for act in mact :
    if act["act"] == "inform" :
        for slot,value in act["slots"]:
            informed_slots.append(slot)

for slot in
set(requested_slot_stats.keys()
+ hyps["requested slots"].keys()):
# if slot not in informed_slots.keys():
    p = requested_slot_stats[slot]

    if slot not
    in hyps["requested slots"].keys():
        hyps["requested slots"][slot]
          = 0.0
    # your code here
    x = p + (1    p)
    * hyps["requested slots"][slot]

    # clip the score
    hyps["requested slots"][slot]
    = clip(x)
self.hyps = hyps
return self.hyps
```

B. Code: Monte-Carlo Control

```
QBest = None
actionBest =
Settings.random.choice(admissible)

for a in admissible:
    dp, _ = self.findClosest(flat_belief, a)

    if not dp in self.dictionary:
        continue

    # Update best
    Q = self.dictionary[dp].Q
    if not QBest or Q > QBest:
        QBest = Q
        actionBest = a
action = actionBest
return action_names[action], action
```

C. Code: Gaussian-Process SARSA

```
if self.terminal:
    g_new = np.zeros(len(self._dictionary))
    delta_new = 0.0
else:
    k_tilda = self.k_tilda(state, action, kernel)
    g_new = np.dot(self._K_tilda_inv, k_tilda)
    delta_new = kernel.Kernel(state, state)*
    kernel.ActionKernel(action, action)
      np.dot(k_tilda, g_new)

## END TODO ##
k_tilda_new = self.k_tilda(state, action, kernel)
_a_new = g_new

delta_k_tilda_new = k_tilda_prev
(np.zeros(len(self._dictionary)) if self.terminal

_d_new = reward +
(0.0 if self.initial else
(self._gamma * (self._sigma ** 2)*
self._d)/self._s ) \
  np.dot(delta_k_tilda_new,
self._alpha_tilda)

self._d =_d_new
```

X. Acknowledgment

The author would like to thank the Dialogue Systems Group at Cambridge University Engineering Department for providing the Dialogue System Toolkit. We would like to thank Pei-Hao (Eddy) Su and Milica Gasic for the enormous support through teaching and practical demonstrations. Finally, we thank the Cambridge University MPhil Machine Learning,

Speech and Language Technology program for providing the practical experimentation opportunity.

REFERENCES

[1] Steve J. Young, Milica Gasic, Blaise Thomson, and Jason D. Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5): 1160–1179, 2013. doi: 10.1109/JPROC.2012.2225812. URL http://dx.doi.org/10.1109/JPROC.2012.2225812.

[2] M. Henderson, B. Thomson, and J. Williams. The Third Dialog State Tracking Challenge. In *Proceedings of IEEE Spoken Language Technology*, 2014.

[3] Matthew Henderson. *Discriminative Methods for Statistical Spoken Dialogue Systems*. PhD thesis, University of Cambridge, 2015.

[4] Matthew Henderson, Blaise Thomson, and Steve J. Young. Robust dialog state tracking using delexicalised recurrent neural networks and unsupervised adaptation. In *2014 IEEE Spoken Language Technology Workshop, SLT 2014, South Lake Tahoe, NV, USA, December 7-10, 2014*, pages 360–365, 2014. doi: 10.1109/SLT.2014.7078601. URL http://dx.doi.org/10.1109/SLT.2014.7078601.

[5] M. Henderson, B. Thomson, and S. J. Young. Deep Neural Network Approach for the Dialog State Tracking Challenge. In *Proceedings of SIGdial*, 2013.

[6] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 116. Cambridge Univ Press, 1998.

[7] Milica Gasic, Catherine Breslin, Matthew Henderson, Dongho Kim, Martin Szummer, Blaise Thomson, Pirros Tsiakoulis, and Steve J. Young. On-line policy optimisation of bayesian spoken dialogue systems via human interaction. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 8367–8371, 2013. doi: 10.1109/ICASSP.2013.6639297. URL http://dx.doi.org/10.1109/ICASSP.2013.6639297.

[8] Milica Gasic, Matthew Henderson, Blaise Thomson, Pirros Tsiakoulis, and Steve J. Young. Policy optimisation of pomdp-based dialogue systems without state space compression. In *2012 IEEE Spoken Language Technology Workshop (SLT), Miami, FL, USA, December 2-5, 2012*, pages 31–36, 2012. doi: 10.1109/SLT.2012.6424165. URL http://dx.doi.org/10.1109/SLT.2012.6424165.

[9] Milica Gasic and Steve J. Young. Gaussian processes for pomdp-based dialogue manager optimization. *IEEE/ACM Trans. Audio, Speech & Language Processing*, 22(1):28–40, 2014. doi: 10.1109/TASL.2013.2282190. URL http://dx.doi.org/10.1109/TASL.2013.2282190.

[10] Jan Peters and J. Andrew Bagnell. Policy gradient methods. In *Encyclopedia of Machine Learning*, pages 774–776. 2010. doi: 10.1007/978-0-387-30164-8_640. URL http://dx.doi.org/10.1007/978-0-387-30164-8_640.

[11] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 387–395, 2014. URL http://jmlr.org/proceedings/papers/v32/silver14.html.

[12] Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors. *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, 2010. ISCA.

[13] Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors. *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014. URL http://papers.nips.cc/book/advances-in-neural-information-processing-systems-27-2014.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[15] Heriberto Cuayhuitl, Supervisors Prof, Steve Renals, and Dr. Oliver Lemon. Spoken dialogue management using hierarchical reinforcement learning and dialogue simulation, 2005.