PARAMETER LEARNING FOR POMDP SPOKEN DIALOGUE MODELS

B. Thomson, F. Jurčíček, M. Gašić, S. Keizer, F. Mairesse, K. Yu, S. Young

Cambridge University Engineering Department

ABSTRACT

The partially observable Markov decision process (POMDP) provides a popular framework for modelling spoken dialogue. This paper describes how the expectation propagation algorithm (EP) can be used to learn the parameters of the POMDP user model. Various special probability factors applicable to this task are presented, which allow the parameters be to learned when the structure of the dialogue is complex. No annotations, neither the true dialogue state nor the true semantics of user utterances, are required. Parameters optimised using the proposed techniques are shown to improve the performance of both offline transcription experiments as well as simulated dialogue management performance.

***Index Terms*—** POMDP, dialogue management, spoken language understanding, expectation propagation.

1. INTRODUCTION

Various researchers have shown that spoken dialogue systems based on the partially observable Markov decision process (POMDP) outperform standard alternatives [1, 2, 3]. The POMDP model views the dialogue as a sequence of turns where the state of the dialogue at each turn is a hidden random variable. The system must estimate its beliefs about the dialogues state using standard probability theory. The probability distributions that model this hidden state are called the *user model*. It is important that the user model's parameters are chosen appropriately.

The most common approach used by previous researchers in selecting these parameters has been to hand-craft them [4, 5, 3]. This can be time-consuming, requires domain knowledge and may be sub-optimal. A more appealing idea is to try to learn the parameters from data. In [6], maximum likelihood estimates are used, but this requires one to annotate the dialogue state. There are many situations where this is impractical or even impossible, since the true dialogue state may be unknown, even to a human annotator. [7] has shown how expectation maximisation (EM) can be used, but that research requires the user's goal in the dialogue to remain constant. This is an unsuitable assumption for many real-world dialogues. EM has also been used to learn a complete user model for a small system (7 states) [8], but this approach does not scale well.

Recently, a method was described for optimising the model parameters based on the reward obtained in a dialogue [9]. While the reward is a good metric for optimising dialogue management performance the approach requires many million online dialogues and also assumes a mechanism for determining the reward. These requirements are not always feasible with human users.

This paper describes an alternative approach for learning the user model that requires only a small corpus of dialogues and can be trained offline. A reward is required only to learn the dialogue policy. The approach can be used in a completely unsupervised setting, requiring no human annotations, and can be trained on simulated or human dialogues. The approach also has the advantage that the parameters can be used for other tasks besides dialogue management, for example in transcription tasks.

The proposed method is tested in two ways. First, two POMDP-based dialogue managers are trained for the same task but using different user model parameters. One of these user-models is hand-crafted while the other is trained on the true semantics of the user utterances. The POMDP system with optimised user model parameters is shown to outperform the system with hand-crafted parameters. Second, a user model with optimised parameters is used to recompute the probabilities of an N-best list of semantics in a corpus of human-machine dialogues. The semantics are represented as user *dialogue acts*, which are an abstract representation of the user's intent in an utterance. The recomputed N-best lists are shown to be more accurate when using trained parameters (when trained on either hypothesised user dialogue acts or annotated user dialogue acts).

The paper is organised as follows. Section 2 presents the background theory used in the paper and describes the structure of the dialogue model used. This section includes a description of an example user model for modelling tourist information dialogues. The expectation propagation (EP) algorithm is described in section 3. EP provides an elegant method for extending the belief propagation algorithm to learning the parameters of the user model. In section 4, the learned parameters are used to improve dialogue performance and section 5 describes how the learned user model parameters can improve the transcription of user's dialogue acts. Section 6 concludes

This work was supported by the EU FP7 CLASSiC project. Email addresses: {brmt2, fj228,mg436,sk561,farm2,ky219,sjy}@eng.cam.ac.uk

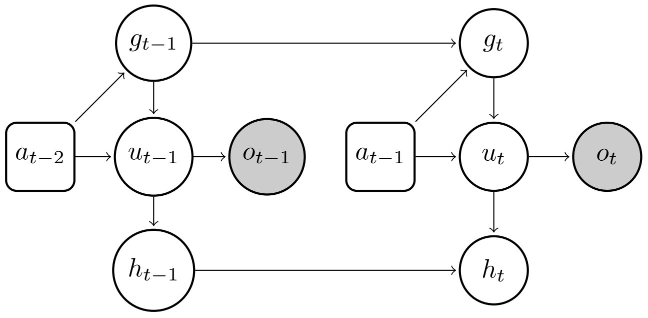

Fig. 1. Dependencies in the BUDS dialogue system.

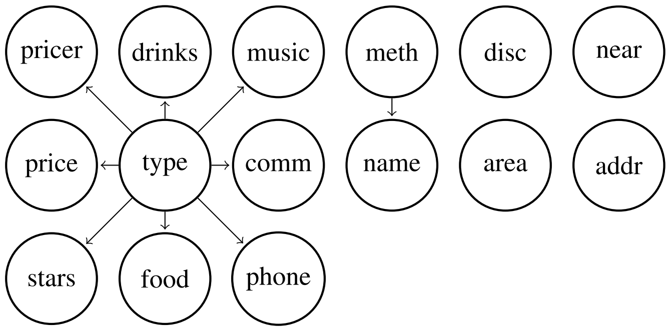

Fig. 2. Sub-goals in the TOWNINFO system.

the paper.

2. THE DIALOGUE MODEL

The POMDP model of dialogue assumes that a dialogue consists of a sequence of turns, t. In each turn, the dialogue state, s_t, is hidden and depends on the previous state, s_{t-1}, and the system's previous action, a_{t-1}. The system's beliefs about the dialogue state, b_t, is a probability distribution over the state at time t, s_t. These beliefs are updated based on observations the system receives about the user, o_t, where o_t depends on s_t and a_{t-1}. In this paper, the observation is represented as an N-best list of user dialogue acts with associated confidence scores.

This paper makes use of a special case of the POMDP model, called the Bayesian update of dialogue state (BUDS) model [3]. In the BUDS model, the dialogue state is further factorized into a user goal, g_t, the user's true dialogue act, u_t, and a history of the dialogue h_t. The user goal and dialogue history are further factorized into a series of *slots*, i, with slot-level goals (called *sub-goals*) labelled $g_{t,i}$ and slot-level histories denoted $h_{t,i}$. Conditional independence assumptions are taken such that the $g_{t,i}$ depend on a_{t-1} and $g_{t-1,i}$ and optionally a parent goal, $g_{t,pa(i)}$. u_t depends on a_{t-1} and the $g_{t,i}$ and $h_{t,i}$ depend on $h_{t-1,i}$ and u_t. The dependencies between the unfactorized variables are shown in Figure 1.

2.1. TOWNINFO : An example system

The TOWNINFO system, which will be used in the later experiments, provides a good illustration of this framework. The system's task is to provide tourist information about hotels, restaurants and bars in a fictitious town. The user may constrain their requests according to nine slots : type of venue, type of food, number of stars, area, price range, type of music, type of drinks, nearness to a particular venue and venue name. The system has four information slots, where the user cannot constrain the value but can ask for information: phone number, address, comment and price. Finally, a further two slots are used for modelling other features of the dialogue: one (`disc`) handles discourse actions such as the user asking to repeat or restart; the other (`meth`) models the style of interaction (for example, asking for a venue matching some constraints, asking for alternatives). Figure 2 presents a Bayesian network representation of the sub-goals in the TOWNINFO system. The dependencies shown are constraints so that a child goal is only applicable when the parent variable has a suitable value. For example, food is only applicable when the type is "restaurant".

2.2. The parameters and probability factors

The BUDS dialogue model makes use of several probability distributions, each with its own conditional independence assumptions. It is important to realise that these probability distributions will depend on some parameters, Θ. These parameters determine the probability for each possible output value according to the definition of the relevant probability factor. Three forms of probability factor are used in the TOWNINFO system. Two of these forms model the evolution of sub-goals, while the third models the probability of user acts given a collection of sub-goals.

The evolution of all sub-goals depends on how the previous system action, a_t, relates to the previous sub- goal, $g_{i,t-1}$, and also on whether the parent sub-goal, $g_{pa(i),t}$, is valid. Sub-goals with no parent are modelled as sub-goals where the parent is always valid. In the TOWNINFO system, twelve different *parent categories* are defined, denoted by ρ. These differentiate, for example, between: (1) when the system has said that no venue with a particular slot-value exists, (2) when the system has not informed a venue and (3) cases where the parent is not applicable. The special value $\rho =$"N/A" is used to denote that the parent sub-goal is not applicable. A complete description of the categories is provided elsewhere [10].

One can now define the two types of goal factors. The first models how the `meth` and `disc` goals evolve over time. The changes in these goals are unconstrained, so a probability is required for every possible pair of values. For each of the slots, i, each parent category, ρ, and every possible value j, k, a parameter, $\theta_{i,\rho,j,k}$ determines the probability

$$p(g_{t,i} = j | g_{t-1,i} = k, \rho) = \theta_{i,\rho,j,k}.$$

The second type of factor models the evolution of all remaining goals, with parameters tied to enforce a constant probability of change. All slots include a special "N/A" value for when the parent is not applicable and a "dontcare" value for when the user doesn't mind. For each of slot, i, each parent category, ρ, and $l, m \in \{$"N/A", "dontcare", "value1", "value2"$\}$, a parameter, $\theta_{i,\rho,l,m}$ is defined, along with counts n_i and n_i'. n_i equals the number of non-special values while n_i' equals the number of values excluding "N/A" (but including "dontcare"). Given a slot-value j, the value of l is selected as "N/A" or "dontcare" if j is one of these and "value1" otherwise. Given slot-values j, k the value of m is selected as "N/A" or "dontcare" if k is one of these, "value1" if $k = j$ and "value2" otherwise. The basic form of the probability factor $p(g_{t,i} = j | g_{t-1,i} = k, \rho)$ is then defined as

$$\begin{array}{cc} 0 & \text{if } \rho = \text{"N/A"}, m \neq \text{"N/A"} \\ 1 & \text{if } \rho = \text{"N/A"}, m = \text{"N/A"} \\ 0 & \text{if } \rho \neq \text{"N/A"}, m = \text{"N/A"} \\ 0 & \text{if } \rho \neq \text{"N/A"}, l = \text{"dontcare"}, m = \text{"value1"} \\ \frac{\theta_{i,\rho,l,m}}{n_i'-1} & \text{if } \rho \neq \text{"N/A"}, l = \text{"dontcare"}, m = \text{"value2"} \\ \frac{\theta_{i,\rho,l,m}}{n_i-1} & \text{if } \rho \neq \text{"N/A"}, l \neq \text{"dontcare"}, m = \text{"value2"} \\ \theta_{i,\rho,l,m} & \text{otherwise} \end{array}.$$

A similar, though slightly more complex, form is used when the system informs about venues [10]. This is enabled by adding special cases into the parent category ρ for "system informed a venue with this value" and "system informed a venue with different value". This extension does not affect the basic structure of the factor.

The final factor to be defined is the probability of a user action given the sub-goals and system action, $p(u|\{g_i\}_i, a)$, where $\{g_i\}_i$ denotes the collection of sub-goals. Given the collection of variables, hand-crafted rules are used to determine whether the user act contradicts each of the sub-goals, with the contradiction state denoted $c_{u,g_i,a} \in \{$"contradicts", "no contradiction"$\}$. The collection of slots appearing in the act is denoted by I_u. Three parameters $\theta_{*,0}, \theta_{*,1}, \theta_{*,2}$ are defined with the probability factor $p(u|\{g_i\}_i, a)$ proportional to

$$\begin{array}{cc} \theta_{*,0} & \text{if there exists } c_{i,g_i,a} = \text{"contradicts"} \\ \theta_{*,1} & \text{if } u = \text{"silence()"} \\ \theta_{*,2} \prod_{i \in I_u} n_i & \text{otherwise} \end{array}.$$

The factor is defined in this way so that the user model can distinguish between invalid acts given a goal, the user keeping silent, and sensible user actions. It is expected that $\theta_{*,0}$ is close to zero, although a small positive value is permitted to allow for user errors. The $\prod_{i \in I_u} n_i$ factors are used to reduce the bias due to user acts having different prior probabilities of contradicting the goals. Without this factor, the probability of a user act not contradicting would constantly decrease as more slots are spoken about. For example, the prior probability of the user act "inform(food=Chinese, price range=cheap)" would always have a lower value than for "inform(food=Chinese)" because the latter user act will contradict less goals than the former.

This paper will present a Bayesian approach to learning the parameters for all these factors. As such, a collection of priors is defined for the parameters. The parameters are grouped together into vectors, where the final index on the parameter denotes the index in the vector. The parameter vectors are therefore denoted $\boldsymbol{\theta}_{i,\rho,j}$, $\boldsymbol{\theta}_{i,\rho,l}$ and $\boldsymbol{\theta}_*$. The prior distributions used for each of these parameters are Dirichlet distributions with parameters $\boldsymbol{\alpha}^p_{i,\rho,j}$, $\boldsymbol{\alpha}^p_{i,\rho,l}$ and $\boldsymbol{\alpha}^p_*$. The Dirichlet distribution for variable $\boldsymbol{\theta}$ with parameters $\boldsymbol{\alpha}$ is the distribution

$$p(\boldsymbol{\theta}) = \frac{\Gamma(\sum_j \hat{\alpha}_j)}{\prod_j \Gamma(\hat{\alpha}_j)} \prod_j \theta_j^{\alpha_j - 1},$$

where $\Gamma(z)$ is the gamma function,

$$\Gamma(z) = \int_0^\infty t^{z-1} \exp(-t) dt.$$

3. EXPECTATION PROPAGATION

When all the nodes in a Bayesian network are discrete, one can use the belief propagation algorithm to calculate the marginal distributions of the random variables. The user model parameters are drawn from a continuous space and so this standard algorithm cannot be applied to learning them. Expectation propagation [11] is an extension of belief propagation to continuous variables and provides a suitable approach to estimating these parameters.

The joint distribution of all variables, X, in the Bayesian network can be written as a product of probability factors, $p(X) = \prod_f p_f(X)$, with f indexing the factors. Each factor gives the probability of a variable given its parents in the Bayesian network. The collection of variables linked to factor f is denoted X_f. The joint can therefore be written $p(X) = \prod_f p_f(X_f)$. When a collection of variables is observed, the joint posterior distribution is again proportional to $\prod_f p_f(X_f)$, with observed variables replaced by their observed value. Expectation propagation attempts to find an approximation to this posterior, $q(X) = \prod q_f(X_f) \approx p(X)$. This paper uses a factorized approximation where each factor is further factorized: $q_f(X_f) = \prod_j q_f(x_j)$, with j indexing all variables and $q_f(x_j)$ constant for variables not appearing in the factor.

EP solves for this approximation one factor at a time. A particular factor, $\tilde{f}$, is chosen and all other factors are fixed. One must then find $q_{\tilde{f}}(X_{\tilde{f}}) = \prod_j q_{\tilde{f}}(x_j)$ to minimize $KL(p||q^{\backslash\tilde{f}} q_f)$, where

$$q^{\backslash\tilde{f}}(X_{\tilde{f}}) \propto \prod_{f \neq \tilde{f}} q_f(X_f).$$

The function $q^{\backslash \tilde{f}}(X_{\tilde{f}})$ denotes the *cavity distribution*, obtained by multiplying all approximations except for $\tilde{f}$. The cavity distribution as a function of a single variable x_j is similarly defined as

$$q^{\backslash \tilde{f}}(x_j) \propto \prod_{f \neq \tilde{f}} q_f(x_j). \quad (1)$$

The function $q^{\backslash \tilde{f}} q_{\tilde{f}}$ is called the *target function*.

3.1. EP for a discrete probability factor

In the case of this paper, all the probability factors to be approximated give the probability of a discrete output variable given a collection of discrete input variables, and a collection of parameter vectors. The case of the unconstrained factor, as described for the `meth` and `disc` sub-goals in section 2.2, is presented here. For clarity, the effect of the parent category, ρ, and the index of the slot, i, are omitted.

The chosen probability factor, $\tilde{f}$, has the form

$$p(g_t = j | g_{t-1} = k) = \theta_{j,k}.$$

One must obtain approximating functions $q_{\tilde{f}}(g_t)$, $q_{\tilde{f}}(g_{t-1})$, and $q_{\tilde{f}}(\boldsymbol{\theta}_j)$. All $q_f(\boldsymbol{\theta}_j)$ approximations are constrained to the Dirichlet distribution, with the parameters denoted by $\boldsymbol{\alpha}_{f,j}$. The approximations for other factors are fixed and the cavity distributions for the variables are defined as per equation 1. In the case of the discrete variables g_t and g_{t-1}, the cavity distributions are computed by multiplying all factor approximations except for $\tilde{f}$. The cavity distribution for the parameters $\boldsymbol{\theta}_j$ is a product of continuous distributions. For N_f different factors, the cavity distribution is the Dirichlet distribution with parameters

$$\boldsymbol{\alpha}_j^{\backslash \tilde{f}} = \sum_{f \neq \tilde{f}} \boldsymbol{\alpha}_{f,j} - (N_f - 1)\mathbf{1}. \quad (2)$$

Note that when the parameters $\boldsymbol{\theta}_j$ do not appear in a factor, the approximation is constant and the vector of approximation parameters, $\boldsymbol{\alpha}_{f,j}$, equals the vector of ones, $\mathbf{1}$.

Given the cavity distributions, one can show that the discrete approximating functions that minimize $KL(p||q^{\backslash \tilde{f}} q_{\tilde{f}})$ are [10]

$$q_{\tilde{f}}(g_t) \propto \sum_{g_{t-1}} q^{\backslash \tilde{f}}(g_{t-1}) \mathbb{E}(\boldsymbol{\theta}_{g_{t-1},g_t} | q^{\backslash \tilde{f}}(\boldsymbol{\theta}_{g_{t-1}})), \quad (3)$$

$$q_{\tilde{f}}(g_{t-1}) \propto \sum_{g_t} q^{\backslash \tilde{f}}(g_t) \mathbb{E}(\boldsymbol{\theta}_{g_{t-1},g_t} | q^{\backslash \tilde{f}}(\boldsymbol{\theta}_{g_{t-1}})), \quad (4)$$

where the expectations are taken over $q^{\backslash \tilde{f}}$.

It can be shown that to minimize the KL divergence, the set of parameters for the target function, denoted $\boldsymbol{\alpha}_j^*$, must satisfy the following equation for every k [10],

$$\Psi(\alpha_{jk}^*) - \Psi(\sum_{l=1}^{N_\alpha} \alpha_{jl}^*) = c_{jk}, \quad (5)$$

where N_α denotes the number of values,

$$c_{jk} = \Psi(\alpha_{jk}^{\backslash \tilde{f}}) - \Psi(\sum_{l=1}^{N_\alpha} \alpha_{jl}^{\backslash \tilde{f}}) + \frac{w_{jk}}{\alpha_{jk}^{\backslash \tilde{f}}} - \frac{1 - w_{j0}}{\sum_{l=1}^{N_\alpha} \alpha_{jl}^{\backslash \tilde{f}}}, \quad (6)$$

$\Psi(z)$ is the digamma function,

$$\Psi(z) = \frac{d}{dz} \log \Gamma(z), \quad (7)$$

and the w_{jk} are weights ($\sum_k w_{jk} = 1$):

$$w_{j0} \propto \sum_{j' \neq j} q^{\backslash \tilde{f}}(g_{t-1} = j'), \quad (8)$$

$$w_{jk} \propto q^{\backslash \tilde{f}}(g_{t-1} = j) q^{\backslash \tilde{f}}(g_t = k) \frac{\alpha_{jk}^{\backslash \tilde{f}}}{\sum_{l=1}^{N_\alpha} \alpha_{jl}^{\backslash \tilde{f}}}. \quad (9)$$

Various methods are possible for solving equation 5. The approach used here is taken from Section 3.3.3. of [12]. Let $\Delta = \Psi(\sum_{k=1}^{N_\alpha} \alpha_{ik}^*)$, and make α_{ij}^* the subject of the formula in equation 5,

$$\alpha_{jk}^* = \Psi^{-1}(c_{jk} + \Delta). \quad (10)$$

Summing over k and taking both sides as arguments for the Ψ function gives,

$$\Delta = \Psi(\sum_{l=1}^{N_\alpha} \alpha_{jl}^*) = \Psi\left(\sum_{l=1}^{N_\alpha} \Psi^{-1}(c_{jl} + \Delta)\right). \quad (11)$$

One can now solve for Δ using Newton's method and use equation 10 to obtain the $\boldsymbol{\alpha}_j^*$ parameters. The desired approximating function parameters are then calculated as

$$\boldsymbol{\alpha}_{\tilde{f},j} = \boldsymbol{\alpha}_j^* - \boldsymbol{\alpha}_j^{\backslash \tilde{f}}. \quad (12)$$

The full algorithm operates by repeatedly choosing a factor to update, computing the cavity distributions in terms of the current approximations (equations 1 and 2) and then updating the current approximating functions as per equations 3,4 and 12. Similar to belief propagation, the process is repeated until changes in the approximating functions are below a threshold. The other forms of probability factor are similar [10].

4. DIALOGUE MANAGEMENT EVALUATION

An analysis of simulated dialogue management performance is the first method used here for evaluating the proposed method. To this end, an agenda-based user simulator was built for the TownInfo task [13]. The simulator uses handcrafted, probabilistic rules to simulate the dialogue act that a user would respond with in a given situation. This dialogue act is then passed to an error-simulator which simulates how the dialogue act would be confused and how the confidence scores would be generated.

The error simulator outputs an N-best list of user dialogue acts with associated confidence scores (in these experiments $N = 3$). The simulator is parameterized by an error rate, r, and a variability parameter V. Given these values, the confidence score parameter vector, $\boldsymbol{\alpha}_r$ of size $N + 1$ is defined by

$$\boldsymbol{\alpha}_r{}^\top = \left(Vr, \frac{V(1-r-r^2)}{N-1}, \ldots, \frac{V(1-r-r^2)}{N-1}, Vr^2 \right).$$

For each turn, a vector of confidence scores is drawn from the Dirichlet distribution with parameters $\boldsymbol{\alpha}_r$. These confidence scores are used as probabilities to draw a position between 1 and $N + 1$. The true user act is placed at this position in an $N + 1$-best list. All remaining positions are assigned a confused user act, using hand-crafted rules to alter it. The item at position $N + 1$ is dropped and the list is passed to the dialogue manager.

Using the above simulation environment, a policy was trained with the natural actor critic algorithm, as described in [3]. The initial system was trained using a collection of hand-crafted user model parameters, and is denoted by HDC-USER. These parameters were hand-tuned by the system designer in an attempt to improve dialogue performance and had been used in previous experiments. 1000 dialogues with the system were then simulated with a hand-crafted policy and the true semantics in these dialogues were used to train a user model as described in section 3. The resulting policy is denoted by TRN-USER. In both cases 800,000 dialogues were simulated at an error rate of $r = 0.4$ to train the policy.

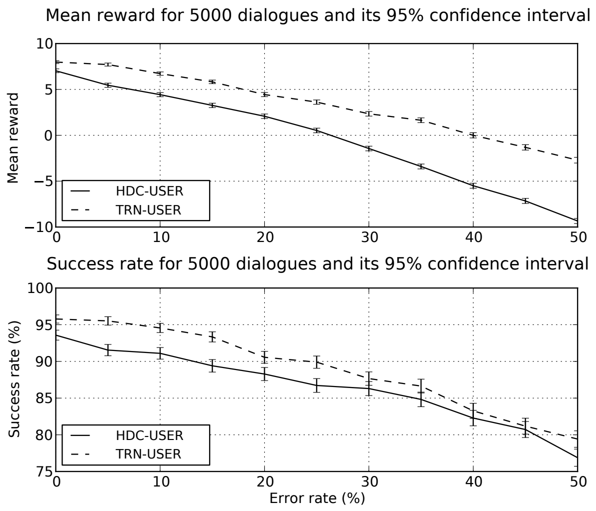

Fig. 3. The effect of user model parameter learning on dialogue manager performance.

Figure 3 shows a performance comparison between the resulting dialogue managers. The dialogue managers are compared at different simulated error rates, with estimated performance computed on 5000 dialogues at each error rate. For a dialogue of length T, the reward is computed as $20 - T$ if the dialogue is successful and $0 - T$ otherwise. The figure shows that the system with trained user-model parameters significantly outperforms the alternative at all error rates, in terms of both success and reward.

5. SEMANTIC RE-SCORING EVALUATION

A second application of the learned POMDP user model is its use in transcribing human-machine dialogues. The effectiveness of training the user-model for this task will be evaluated on both simulated and human-machine dialogues. In each case, the N-best list of dialogue acts is added as an observation to the Bayesian network. When EP is run, the updates of factors connected to the user act will result in a collection of recomputed probabilities for them. This can be used as a re-scored hypothesis list.

The intuition here is that the user model provides a mechanism for incorporating evidence from different stages of the dialogue when transcribing other parts. As an example, a user who wants a restaurant at the beginning of a dialogue is unlikely to change until they are offered something suitable.

The first re-scoring experiment makes use of the simulated environment described above. 1000 dialogues are simulated and used to train the user model. A further 1000 dialogues are simulated, and the confidences on the hypothesised semantic N-best lists are recomputed by running EP. The accuracy of the top hypothesis (TAcc), item-level normalized cross entropy (NCE), oracle accuracy (OAcc) and item-level cross-entropy (ICE) metrics are then computed [10]. TAcc, OAcc and NCE show improvements as increases in the metric value while ICE shows improvements as decreases in the metric value. Table 1 shows the effects of re-scoring with user models trained from the hypothesised semantics (LEARNED (NOISY)) and from the true user acts (LEARNED).

	OAcc	**TAcc**	**NCE**	**ICE**
NO RE-SCORING	93.5	75.7	0.586	0.921
LEARNED (NOISY)	93.5	**81.7**	**0.650**	**0.870**
LEARNED	93.5	81.5	0.632	0.903

Table 1. Evaluation of semantics after re-scoring simulated data based on hypothesised semantics for the entire dialogue. The simulated data contains 1000 dialogues, with 14358 simulated user utterances.

One can see from the table that the user model improves transcription performance on the simulated data. The accuracy of the top hypothesis increases by 6% absolute, which is 34% of the available increase given the oracle accuracy. The NCE and ICE scores, which evaluate the usefulness of the confidence scores also improve.

A second re-scoring experiment was performed on human-machine dialogues. Two corpora of dialogues in the TOWN-

INFO domain were compiled. The first corpus contains 720 dialogues from 36 different speakers interacting with 5 different systems [14]. This corpus is denoted MAR09. The second corpus contains 648 dialogues from 36 different speakers interacting with 6 different systems [2, 3]. This corpus is denoted FEB08. Both corpora contain annotations of the true user dialogue act for all turns.

Table 2 presents the results of user model re-scoring on these corpora. Two user models were trained on the MAR09 corpus. The first (LEARNED (NOISY)) uses only the hypothesised semantics for training while the second uses the true user actions transcribed by a human annotator (LEARNED). These user-models were used to re-score the semantics on the FEB08 corpus and the OAcc, TAcc, NCE and ICE metrics were computed on the result.

	OAcc	**TAcc**	**NCE**	**ICE**
NO RE-SCORING	79.2	73.3	-0.033	1.687
LEARNED (NOISY)	79.2	73.4	0.327	**1.586**
LEARNED	79.2	**73.9**	**0.338**	1.655

Table 2. Evaluation of semantics after re-scoring the FEB08 corpus.

The use of the user model on the human-machine data improves the usefulness of the confidence scores as can be seen by the increase in the NCE scores. The accuracy of the top hypothesis does increase, though the increase is small compared to the simulated experiment. This can be attributed to the difference in the type of confusions in the human-machine and simulated data. The biggest increase in accuracy is 0.5% absolute, which is 8.5% of the available increase given the oracle accuracy.

6. CONCLUSION

This paper has described how expectation propagation can be used to learn the user model parameters for a POMDP model of dialogue. The approach requires no annotations of either the dialogue state or the true user dialogue acts. Experiments show that the proposed approach improves dialogue management performance as well as transcriptions of the user's dialogue acts. Future work should show that the improvements in semantic accuracy can extend to improvements in the speech recognition performance, that learning of user model parameters improves performance of dialogues with human users and that the same methods can be used to build a trainable user simulator.

7. REFERENCES

[1] J. D. Williams and S. Young, "Partially observable Markov decision processes for spoken dialog systems," *Computer Speech and Language*, 2006.

[2] M. Gašič, 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.

[3] B. Thomson and S. Young, "Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems," *Computer Speech & Language*, 2009.

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

[5] S. Young, M. Gašič, 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 & Language*, 2009.

[6] J. D Williams, "Exploiting the ASR N-best by tracking multiple dialog state hypotheses," in *Proceedings of Interspeech*, 2008.

[7] U. Syed and J. D Williams, "Using automatically transcribed dialogs to learn user models in a spoken dialog system," *Proceedings of the ACL*, 2008.

[8] F. Doshi and N. Roy, "Spoken language interaction with model uncertainty: an adaptive human-robot interaction system," *Connection Science*, 2008.

[9] F. Jurčíček, B. Thomson, S. Keizer, F. Mairesse, M. Gašič, K. Yu, and S. Young, "Natural belief-critic: a reinforcement algorithm for parameter estimation in statistical spoken dialogue systems," in *Proceedings of Interspeech*, 2010.

[10] B. Thomson, *Statistical methods for spoken dialogue management*, Ph.D. thesis, University of Cambridge, 2009.

[11] T. Minka, "Expectation propagation for approximate Bayesian inference," *Proceedings of UAI*, 2001.

[12] U. Paquet, *Bayesian inference for latent variable models*, Ph.D. thesis, University of Cambridge, 2007.

[13] J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young, "Agenda-based user simulation for bootstrapping a POMDP dialogue system," in *Proceedings of HLT/NAACL*, 2007.

[14] P. Bretier, P. Crook, S. Keizer, R. Laroche, O. Lemon, and G. Putois, "CLASSiC deliverable D6.3: Initial evaluation of CLASSiC TOWNINFO and self-help systems," Tech. Rep., June 2009.