LOG-LINEAR DIALOG MANAGER

Hao Tang∗

Toyota Technological Institute at Chicago
6045 S. Kenwood Ave., Chicago, IL

Shinji Watanabe, Tim K. Marks, and John R. Hershey

Mitsubishi Electric Research Laboratories (MERL)
201 Broadway, Cambridge, MA

Abstract

We design a log-linear probabilistic model for solving the dialog management task. In both planning and learning we optimize the same objective function: the expected reward. Rather than performing full policy optimization, we perform on-line estimation of the optimal action as a belief-propagation inference step. We employ context-free grammars to describe our variable spaces, which enables us to define rich features. To scale our approach to large variable spaces, we use particle belief propagation. Experiments show that the model is able to choose system actions that yield a high expected reward, outperforming its POMDP-like log-linear counterpart and a hand-crafted rule-based system.

Index Terms— Log-linear Model, POMDP, Dialog Manager

1. INTRODUCTION

A current trend in dialog manager research is to model dialog sessions using partially observable Markov decision processes (POMDPs) [1]. After the seminal work of [2] and more than a decade of subsequent research, statistical dialog systems have become a new standard for advanced dialog systems [3, 4, 5]. For recent developments in spoken dialog systems, see [6, 1] and the citations therein.

In a POMDP dialog system, the dialog is represented by means of a set of random variables at each turn of the dialog: an observed variable representing what the user has said, a hidden state variable representing the progress of the dialog so far, and a system action that has to be selected. The POMDP model defines two probabilistic dependencies: the conditional probability of the current state given the previous state and system action, and the conditional probability of the observation given the current state and previous system action.

A reward function specifies, for each turn, a fitness criterion as a function of the state and chosen action for that turn. Given a reward function, it is possible to determine a policy that provides the optimal system action given what is known about the state distribution at the current time. This policy can then be used to generate system actions in the course of a dialog. Selecting system actions is order to maximize reward is called planning.

To have a working system, one also needs to estimate the model parameters that define probabilities in the POMDP. This estimation is called learning. The parameters are typically estimated using a maximum likelihood criterion, rather than using the reward function. For example, a maximum likelihood dynamic Bayesian network (DBN) is used in [7]. A major problem with these approaches is that planning and learning are optimized separately using different criteria. In addition, planning and learning are notoriously difficult optimization problems [8]. This motivates us to design a simple coherent system that has the advantage of a consistent optimization criterion and at the same time is more efficient to optimize.

Our approach is to model the dialog system using a log-linear probability distribution. We call this a log-linear dialog manager. Log-linear distributions have been increasingly used to model sequences since the introduction of conditional random fields [9]. Although log-linear models in general cannot represent all distribution families, their flexible use of feature functions enables them to express a wide family of probabilistic models. In addition, since our model is a Markov chain, we can exploit efficient algorithms for optimization. In particular, we are interested in optimizing the sum of rewards along the time axis. Similar optimizations are well known for other problems [10, 11].

To represent the space of possible states, user actions, and system actions, we use context-free grammars (CFGs), each of which is based on a graph of semantic representations related to the domain of the dialog system. Instead of being simple multinomials, the random variables take values in the space of parse trees generated by the CFGs. This provides a rich structure that allows us to extract a wide range of features, and is reminiscent of [12, 3], which use graphs or rules to partition the space. Because of the flexible use of features inherent in log-linear models, we can design features that make our dialog system behave like a deterministic rule-based dialog system as a special case. This is done by implementing the rules of the deterministic dialog system as indicator-function features, and initializing the parameters such that the log-linear probability distributions correspond to these rules. As we obtain more data, the learning algorithm can provide a smooth transition from a rule-based system to a statistical data-driven system, which is a desirable feature in real-world scenarios [13].

A common problem in dialog management is that inference becomes intractable in variable spaces large enough to handle real problems. This problem has been addressed in a variety of ways [14, 15, 16, 17]. With our model the optimization can be solved using belief propagation; we employ particle belief propagation [18] to scale the algorithm to large variable spaces.

2. MODEL DEFINITION

Our probabilistic model has four variables at each time step. Two are observable variables: the system action at the observation at time t−1. The other two are latent variables which must be inferred: the user action ut and the state st. Roughly speaking, each step of the dialog proceeds as follows. Based on all of the system actions and observations up to time t−1, the system prompts the user with query at−1. The user’s response to that query is represented by ot (in our system, ot is a sequence of words uttered by the user). The meaning of that response is represented by the user action, ut, which may be inferred from the observation. The new state, st, may be inferred based on the
Fig. 1. The factor graph representation for the distribution in (1).

The aforementioned system action $a_{t-1}$ and user action $u_t$, as well as the previous state $s_{t-1}$. In our system, the state $s_t$ represents the user’s intention, although in general it could also include additional contextual information. Using subscripted colons to denote sequences (e.g., $s_0:t \equiv \{s_0, s_1, \ldots, s_T\}$), an entire dialog session of length $T$ is represented by four variable sequences: $s_0:T$, $a_0:T$, $a_1:T$, $u_1:T$.

Our model for a dialog session is represented by the factor graph in Fig. 1, which for our log-linear model corresponds to the following joint probability distribution over the variables:

$$p(s_0:T, a_0:T, u_1:T, o_1:T) =$$

$$\frac{1}{Z_\theta} \exp \left[ \sum_{t=0}^T \theta_f \phi_f(s_t, a_t, s_{t+1}, u_{t+1}) + \sum_{t=1}^T \theta_g \phi_g(u_t, o_t) \right],$$

where $Z_\theta$ is a normalizing constant, $\phi_f$ and $\phi_g$ are vectors of feature functions, and $\theta_f$ and $\theta_g$, respectively, are vectors of the corresponding model parameters. (Note that at time $t = T$, $s_{t+1}$ and $u_{t+1}$ are undefined, so as shown in factor $f_T$ of the factor graph in Fig. 1, at time $t = T$ we define $\phi_f$ as a function of only its first two inputs.) To simplify notation, we also define the following vectors:

$$\theta \equiv \begin{bmatrix} \theta_f \\ \theta_g \end{bmatrix}, \quad \phi(t) \equiv \begin{bmatrix} \phi_f(s_t, a_t, s_{t+1}, u_{t+1}) \\ \phi_g(u_t, o_t) \end{bmatrix}.$$  

These enable us to rewrite (1) more succinctly as

$$p(s_0:T, a_0:T, u_1:T, o_1:T) = \frac{1}{Z_\theta} \exp \left[ \sum_{t=0}^T \theta^\top \phi(t) \right],$$

where

$$Z_\theta = \sum_{s_0:T, a_0:T, u_1:T} \exp \left[ \sum_{t=0}^T \theta^\top \phi(t) \right].$$

2.1. Variable Spaces

We let $S$, $U$, $A$, and $O$ represent the variable spaces (the set of all possible values) for the variables $s_t$, $u_t$, $a_t$, and $o_t$, respectively. Each observation $o \in O$ can in theory be anything from waveforms or acoustic features to recognized texts. In this work, we let $o \in O$ represent the input word sequence, and we define the variable space $O$ as the set of all sequences of words in a vocabulary set $\mathcal{V}$.

We define each of the variable spaces $S$, $U$, $A$, and $O$ using a context-free grammar (CFG), which consists of a set of production rules. Each variable space is defined as the set of all possible parse trees that can be generated by its CFG. Fig. 3 shows some of the production rules in the CFG that defines the variable space $S$. Each parse tree in $S$ is a possible value of the state $s_t$. Fig. 2 shows one possible value for state $s_1$ (one parse tree in $S$), which was generated using the production rules shown in boldface in Fig. 3.

2.2. Features

As can be seen in the factor graph in Fig. 1 and in (1), there are two types of factors in our model. The first, denoted $f$, models statistical dependencies between the previous and current state, the system action, and the user action. The second, denoted $g$, models dependencies between observed word sequences and their semantic interpretations. For the variables whose spaces are defined using CFGs, we treat each variable value (each parse tree) as a set of active production rules. For example, the production rules that are active in the parse tree of Fig. 2 are highlighted in bold.
other type of feature typically seen in conventional dialog systems is
\[ h_{s:t-1, k' \in S} j \in G_{a-1} \] which additionally requires that production
rule \( j \in G_{a-1} \) is active in system action \( a_{t-1} \). This feature indicates
that a particular system action tends to induce a particular state
transition.

3. PLANNING AND LEARNING

The two basic problems a dialog manager needs to solve are
planning and learning. We assume there is a reward function
how we do planning and learning in terms of the reward function.

3.1. Planning

Planning at time \( \tau \) is the problem of finding the best system action
\( a_{\tau} \), given the history of all previous system actions \( a_{0: \tau-1} \) and
observations \( o_{1: \tau} \). Suppose the dialog has length \( T \). We define the
planning problem as finding \( a_{\tau} \) to maximize the expected reward

\[ \mathbb{E}_{a_{0:T}, o_{1:T}} \left[ \frac{1}{T+1} \sum_{t=0}^{T} r(s_t, a_t) \mid a_{0:T-1}, o_{1:T} \right] \] .

The expectation is taken over all variables not given: all states, all
user actions, and all future system actions and observations.

The above objective could be optimized exactly by hypothesizing
each action \( a_{\tau} \), computing the expected reward given that action
using the sum-product algorithm, and selecting the term that max-
imized expected reward. However, for ease of implementation and
speed, we instead optimize the objective’s variational lower bound,

\[ \mathbb{E}_{a_{0:T}, o_{1:T}} \left[ \prod_{t=0}^{T} \frac{r(s_t, a_t)}{\gamma_t(T+1)} \right] \mid a_{0:T-1}, a_{1:T} \] ,

obtained from Jensen’s inequality, where the \( \gamma_t \) are variational pa-
rameters such that \( \sum_{t} \gamma_t = 1 \). Although the \( \gamma_t \) could be optimized,
we take them to be uniform, setting \( \gamma_t = 1/(T+1) \), to further sim-
plify the computation.

This product form has the nice property that the reward factor-
izes with time. In other words, (6) can be expanded to

\[ \frac{1}{Z'} \exp \left[ \sum_{\tau=0}^{T} \left[ \theta^\top \phi(t) + \gamma_t \log \left( \frac{r(s_t, a_t)}{\gamma_t(T+1)} \right) \right] \right] ,
\]

where \( Z' \) is the partition function of \( p \) with \( a_{0: \tau-1}, o_{1: \tau} \) given. Now
finding the best \( a_{\tau} \) involves just running a standard sum-product
algorithm on the graphical model with an additional term for the
reward. We first collect beliefs from both ends of the graphical model
sending to time \( \tau \), and finding the \( a_{\tau} \) that maximizes (6). If we
write out the belief propagation explicitly, it becomes the familiar
forward-backward algorithm. For example, the forward message is

\[ m_{s_{t+1}} \left( s_{t+1} \right) \frac{m \left( s_t \right)}{m_{s_{t+1} \rightarrow f_j \left( a_j \right) \mid s_{t-1} \rightarrow f_j} \cdot \exp \left( \sum_{t'=0}^{t} \theta^\top \phi(t') + \gamma_t \log \left( \frac{r(s_{t'}, a_{t'})}{\gamma_t(T+1)} \right) \right) \]

\[ \times \exp \left( \theta^\top \phi(s_t, s_{t+1}, a_t, a_{t+1}) + \gamma_{t+1} \log \left( \frac{r(s_{t+1}, a_{t+1})}{\gamma_{t+1}(T+1)} \right) \right) \] .

Note that averaging over future actions using the sum-product
algorithm is different from conventional POMDP optimization,
which seeks to maximize the reward over future system actions.
It is also possible in our approach to use a max-product algorithm
on \( a_t \) while using sum-product on the other variables, to achieve
maximization over future system actions. However, in our system,
as in [19], the model itself contains a stochastic policy that provides
a predictive distribution over future actions; it remains to be seen
whether there is a benefit in maximizing over future actions.

Our approach reduces the space complexity from \( O(|S| |A|) \) to
\( O(|S|) \). The time complexity is also reduced by the same factor.
This is an exact analog to cost-augmented inference in CRFs with
edge-factored costs [10]. To our knowledge, the product form (6)
has never been discussed in the context of POMDPs except in [19].

3.2. Learning

The learning problem is similar to planning, except that instead of
finding the best action we are interested in finding the best model
parameters. In other words, we want to find \( \theta \) such that the expected
reward,

\[ \mathbb{R} \left( \theta \right) = \mathbb{E}_{a_{0:T}, o_{1:T}} \left[ \frac{1}{T+1} \sum_{t=0}^{T} r(s_t, a_t) \mid o_{1:T} \right] , \]

is maximized given all observations \( o_{1:T} \). Again the expectation is
taken over all variables not given, namely all states, all system ac-
tions, and all user actions.

We use gradient descent to optimize the learning objective. It is
well known that for log-linear models, the gradient for this type of
objective has a nice form. In general, for any utility function \( v(x) \)
and probability distribution of the form

\[ p(x) = \frac{1}{Z} \exp \left( \theta^\top \phi(x) \right) , \]

the derivative of the expected utility is:

\[ \frac{\partial}{\partial \theta} \mathbb{E}_{x}[v(x)] = \mathbb{E}_{x}[\phi(x)v(x)] - \mathbb{E}_{x} [\phi(x)] \mathbb{E}[v(x)] . \]

Note that for each parameter \( \theta_i \) in \( \theta \), the derivative is simply the co-
variance between the corresponding feature \( \phi_i \) and the utility. Thus,
the parameters corresponding to features that are positively corre-
lated with utility will be increased, while those whose corresponding
features are negatively correlated with utility will be decreased.

Applying this to our model gives:

\[ \frac{\partial R(\theta)}{\partial \theta} = \frac{\mathbb{E}_{a_{0:T}, o_{1:T}} \left[ \prod_{t=0}^{T} \phi(t) \right]}{Z} \frac{\mathbb{E}_{a_{0:T}, o_{1:T}} \left[ \sum_{t=0}^{T} r(s_t, a_t) \right]}{Z} \]

\[ - \mathbb{E}_{a_{0:T}, o_{1:T}} \left[ \sum_{t=0}^{T} \phi(t) \right] \mathbb{E}_{a_{0:T}, o_{1:T}} \left[ \sum_{t=0}^{T} r(s_t, a_t) \right] , \]

where expectations are computed using \( p(s_{0:T}, a_{0:T}, u_{1:T} \mid o_{1:T}) \). In
the general case, it may be hard to calculate these quantities, but in
our case they can be computed efficiently using belief propagation.
This is an exact analog to optimizing empirical Bayes risk in a CRF
3.3. Particle Belief Propagation

Because the variable spaces are too large to marginalize over, we tackle the problem using particle belief propagation [18], which we briefly review here.

Consider a message \( m_{f_t \rightarrow s_{t+1}} (s_{t+1}) \) passing from factor node \( f_t \) to \( s_{t+1} \) by marginalizing over \( s_t, a_t, \) and \( u_{t+1} \):

\[
m_{f_t \rightarrow s_{t+1}} (s_{t+1}) = \sum_{s_t, a_t, u_{t+1}} \left[ m (a_t) m (s_t) m (u_{t+1}) \times \exp (\theta^T \phi (s_t, s_{t+1}, a_t, u_{t+1})) \right].
\]

If we rewrite the sum with importance sampling, we get

\[
m_{f_t \rightarrow s_{t+1}} (s_{t+1}) = \mathbb{E}_{\pi_t} \left[ m (a_t) m (s_t) m (u_{t+1}) \times \exp (\theta^T \phi (s_t, s_{t+1}, a_t, u_{t+1})) \right],
\]

for some sampling distribution \( \pi_t (a_t, s_t, u_t) \) over which the expectation is computed. We can then approximate it with a sum

\[
m_{f_t \rightarrow s_{t+1}} (s_{t+1}) = \frac{1}{N} \sum_{i=1}^{N} \left[ m (a_t) m (s_t) m (u_{t+1}) \times \exp (\theta^T \phi (s_t, s_{t+1}, a_t, u_{t+1})) \right],
\]

over samples \( \{ (s_t (i), a_t (i), u_t (i), s_{t+1} (i)) \}_{i=1}^{N} \). The choice of sampling distribution is discussed in Section 4.

4. EXPERIMENTS

The dataset consists of 694 successful dialog sessions, which are collected from users interacting with an existing rule-based system. Dialog sessions are selected examples in which the existing dialog system performed well. The average number of turns, \( T \), for the dataset is 3.4. In order to better understand how the model works, we conduct the experiments in a restricted setting by using ground-truth texts instead of one-best recognition results as our observations to the model. The language for the system is in Japanese, so we use McCab [20] to segment Japanese characters into terms as a preprocessing step.

The features we use are of the form \( \mathbb{1}_{k \in S_t, k' \in s_{t+1}, k'' \in u_{t+1}}, \mathbb{1}_{k \in a_t, k' \in s_{t+1}, k'' \in u_{t+1}}, \mathbb{1}_{k \in a, k' \in s_{t+1}, k'' \in u_{t+1}}, \mathbb{1}_{k \in a, k' \in s_{t+1}, k'' \in u_{t+1}} \) as discussed in Section 2.2, except that we do not look at production rules that are more than two steps away from the root of the tree in order to avoid overfitting.

The grammars for states are designed according to the functionality of the existing rule-based system. The grammar for system actions additionally contains production rules for requesting actions from the users, and the grammar for user actions contains other additional rules for filling slots and confirming. There are a total of 171 production rules for states, which can generate a total of 1484 distinct parse trees without expanding all the slot variables.

The reward function we use in a given dialog session only depends on the current action \( a_t \) and not on the current state \( s_t \). We define the goal of a successful dialog from the dataset as the last action performed in the dialog session. If the current action matches the goal, then the reward is 1. If the current action matches any actions \( en \) route to the goal, then the reward is 0.5. Otherwise, the reward is 0.01. In this context, it is important to marginalize over the system actions in the learning objective, so that the system can propose its own actions. Otherwise, if all the actions were observed, the objective function would evaluate to a constant.

For comparison, we implement a simple hand-crafted rule-based system in our framework by setting nonzero parameters only for features corresponding to the state transitions of the existing rule-based system. In addition, we construct a POMDP-like model for comparison, by choosing the largest possible subset of our features so that they act as \( p(s_{t+1} | s_t, a_t), p(a_t | s_t), \) and \( p(u_{t+1} | a_t, s_{t+1}) \) to mimic POMDP.

For evaluation, we train using stochastic gradient descent on the learning objective and compute the expected reward on the test set using five-fold cross-validation. Our sampling distribution for particle belief propagation is uniform for each production rule in the grammar. The marginalization of \( O \) for the factor \( g_t \), for simplicity, was assumed to be uniform for all \( t \) and all \( u \in U \). The number of samples used for each variable was 100. The step size for gradient descent was 100. The number of samples and step size were tuned on a small subset of our dataset. The comparison of the two for the last epoch is shown in Fig. 4.

Across the five folds, both log-linear models achieve higher expected rewards than the rule-based system. In addition, using the full feature set almost always outperforms its POMDP-like counterpart.

The results are promising in that the system performance approaches the maximum possible expected reward for the test set. Although these experiments do not test the planning algorithm, they do test the model predictions, demonstrating that they score almost as well on the test set as the ground-truth actions.

5. CONCLUSION

We present a dialog manager based on a log-linear probabilistic model. We use context-free grammars to impart hierarchical structure to our variables and features. A variational bound on the reward function allows us to perform inference with a single pass of a sum-product algorithm. To handle the large hypothesis space, we use a particle belief propagation method that exploits the grammar’s structure. Future work will investigate use of the grammar for efficient re-sampling, as well as maximization over future system actions.
6. REFERENCES


