POMDP-Based Spoken Dialog Management
by Usman Nisar and Justin Payan

End-to-End System Description

Dialog systems typically consist of three major components. The first is an input processing unit, which in turn contains automatic speech recognition and spoken language understanding. The second major component is the dialog manager. The third major component is the output generation unit, which in turn contains natural language generation and text-to-speech audio generation.

We used a dialogue system development framework called Alex[1] to aid in development of the input processing unit and the output generation unit. The automatic speech recognition engine we used was Google's free ASR API. The API returns highly accurate 1-best recognized text, so we simply accepted the 1-best Google ASR output as the user-generated text. This text was then fed to a handcrafted set of rules for spoken language understanding. Alex contains a few example dialog systems, including a system for planning multi-modal transportation routes. This domain is highly similar to our own, and Alex utilizes a complex rule-based SLU module. Thus we simply modified these rules to include the slots for our own domain. Following this step, the user input consisted of one or more structures of the form "user-dialogue-act(slot-name="value")" or "user-dialogue-act()" with no arguments if appropriate. A list of possible user dialog acts is listed in our presentation.

A POMDP model for dialog management is described in the next section. The POMDP model outputs a single action selected from the list of system dialog acts (a list of system dialog acts is reprinted in our presentation). However, some of the output system dialog acts had to be modified slightly. As an example, the "expl-conf()" act, short for "explicit confirmation", had to be updated with the slot-name and value most recently filled by the user, to produce "expl-conf(slot="slot")&expl-conf(value="value")".

We modified the natural language generation module used by Alex's public transport dialog system to fit our own domain. This module consisted of simple template-based generators. As an example: expl-conf(slot={slot})&expl-conf(value={value}) -> "You want your {slot} to be {value}, is that right?"

Finally, the free VoiceRSS Text-to-Speech API converted the natural language text to an audio file which could be played on the speakers.

POMDP Parameters

Actions are the system dialog acts. States are the user dialog acts, along with the slot names. However, slot values are not included in the states or actions. Leaving out the slot values (and sometimes even the slot names are left out) results in what is called the "summary space". It is common to map into the summary space for the POMDP dialog manager to be computationally feasible. For both actions and states, we limited the number of user/system dialog acts to 1, to avoid combinatorial explosion of the number of actions and states. This is not necessarily always the case in the literature. The initial belief vector was set to have 0.5 probability on the "Initial" state, and uniformly distributed the rest of the probability density among the other 26 states. We tested multiple initial belief distributions, but this distribution seemed best to us.

We set the observation space to be equivalent to the state space, an approach which is common in the literature. However, we made the observation function such that states do not translate to their equivalent observations with probability 1, but more like 0.8. This definition of the observation function captures errors in the ASR model as well as in the SLU model.

We defined the transition probabilities as $p(s_{n+1}|s_n, a_n) = \frac{c(s_n, a_n, s_{n+1})}{c(s_n, a_n)}$, where $c(x)$ simply counts the occurrences of x in the DSTC1 transcribed dialogs.

The reward function was set to -1 for any transition to a non-terminal state, and 10 for any transition to the terminal state ("bye()").

We used APPL's SARSOP solver to solve the POMDP.

[1] https://github.com/UFAL-DSG/alex

Challenges

We found that much of the recent literature talks about complicated factored POMDP models, such as the Hidden Information State POMDP or the Bayesian Update of Dialog State (BUDS) model. Such models incorporate user goal, user action (which we utilized!), and dialog history in the factored state space. These models also tend to represent user and dialog actions as combinations of one or multiple of the system or user dialog act structures described earlier. Both the factored state space, as well as the admission of arbitrarily long compound dialog acts results in a combinatorial explosion in the size of the corresponding POMDP. We could have integrated such models with Alex's NLP input and NLG output modules with minimal thinking (but perhaps large amounts of NLG template writing). However, we determined that solving these POMDP models was out of scope for this project. Thus we restricted our system to only allow single user dialog acts to be considered POMDP states, and only allow single system dialog acts to be considered POMDP actions. We also ignored user goal and dialog history.

The dialogs did not typically end successfully in the terminal "bye()" state, which meant that the reward of 10 was very sparse in the training data. This problem appears to stem from the users getting frustrated with poor system performance and hanging up. Perhaps a human-to-human or simulated-user-to-dialog-system transcript set would provide a POMDP model with more well-motivated parameters.

Although we initially intended to evaluate the system on the metrics originally reported in the DSTC1 release, we determined that we had modified the states too significantly to perform meaningful evaluation in a timely manner. It is interesting to note that although state estimation is an important subtask of the full spoken dialog system, it takes a good deal of manipulation to integrate the state estimation into the dialog system. In general, we found that most of the subtasks in our dialog system were formulated slightly differently as a part of the cohesive system than they would be in isolation. This difficulty in merging isolated components into a seamless dialog system (and the reverse - removing individual components from the dialog system to use them for other applications) should be addressed by future research, if it has not been addressed already. We also need software packages that make these systems more flexible.