Socrates Sim: A User Simulator to Support Task Completion Dialog Research

Dhairya Dalal

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

August 2018

Abstract

The main objective of this project is to ...

Contents

Chapter 1: Introduction

Task completion dialog refers to the space of dialog activities, in which a user engages with an interlocutor in an attempt to complete a task or achieve a tangible goal. For example, imagine a user interacting with a concierge in order to identify and book a restaurant for dinner. In dialog systems, the human interlocutor is replaced by an artificial agent (system/ dialog agent) that can intelligently respond and help the user achieve their goal . One of the key challenges in this space is developing quality and diverse training datasets to support development of dialog agents, many of which rely on data intensive deep learning techniques. Current data gathering practices rely on using crowd-sourced and manual efforts, which are not sustainable and scalable. The virtual simulator attempts to alleviate this need, by simulating a user and generating user utterances in place of a real human user. The user simulator can be used in the context of supervised learning (SL) or reinforcement learning (RL) to train a dialog system to identify optimal dialog policies. There has been significant progress in the dialog and AI space that led to the development of commercially viable intelligent voice systems. In particular, the popularity voices assistants like Amazons Alexa, Apples Siri, and Google assistant have increased the demand for voice interfaces to popular applications and services. There has been a boom in the development of chatbots, third party voice skills for Alexa and Google Home, and proliferation of exciting research applying deep learning and machine learning to the dialog domain. From a research point of view, the dialog domain is

rich, complex, and challenging. Automated dialog systems are not a new concept. The have been used to support call routing (e.g. the press 1 to reach sales) in the context of customer support for banks, credit cards, flight booking, and many other commercial sectors since the 1970s. Central to any dialog system is the dialog policy, which is responsible for informing the system on what to say and what information to collect based on the state of conversation. Traditionally dialog policies were scripted out, usually following a simple flowchart like structure. This is known as a rules based approach, where rules are written out to capture system behavior under predefined situations/states. Rules based systems are limited, as they required the user to follow a scripted paths and provide the system with one piece of information at time. With the advent of personal assistants like Siri and Alexa and chatbot, dialog system are evolving to support wider use cases and more open conversational dialog flows.

However, developing good voice interfaces and systems is still very challenging. For example, a significant percent of the voice skills in the Alexa skill store have poor ratings Rey (2017). According to research by recode.com, 69

Rules based dialogs are not scalable or optimal for more complex dialogs. Researchers have moved to leveraging supervised learning (SL) methods to train dialog systems and produce more robust dialog policies. In an SL approach, the dialog policy is trained to imitate observed actions of an expert using an annotated and manually crafted datasets based real human interactions Schatzmann et al. (2006). While this approach produces better policies than a rules based approach, it is limited to quality and scope of the training data. Producing deep annotated dataset is time consuming, expensive, and the dataset may not comprehensive cover all possible states in a policy space.

As result, reinforcement learning (RL) methods are gain popularity. Given a reward function, the agent can optimize a dialog policy through interaction with

users and learn what an optimal dialog policy should be. As mentioned above, real interaction with users is time consuming and expensive. A user simulator that accurately simulates a real user can allow the RL agent to explore trajectories that may not have existed in observed data and produce larger datasets. The user simulator provides a useful starting point to train and RL based agent, which can be then further optimized in RL situation with real users Li et al. (2016). Currently, there is no general user simulator tool that researchers can use to develop dialog agents for various task completion domains. The aspiration of this thesis is to develop a solution that can fill that gap.

1.1. Prior Work

The growing popularity of statistical approaches for spoken dialog systems has led to research for more optimal ways to generate training dialog data. Supervised learning methods and reinforcement learning methods offer great promise for development of robust goal-oriented task-completion dialog systems. Schatzmann and Young introduced the concept of the hidden agenda user simulation model [6], which has been foundational to conceptualizing user simulators. In their 2009 paper, Schatzmann and Young provided a formalized framework to capture user intents in stack-like structure of pending dialog acts. Bordes & Weston (2016) applied deep learning and neural models to dialog systems. They introduced a network-based end-to-end trainable dialog system, which treated dialog system learning as the problem learning to map dialog histories to systems responses and applying encoder-decoder models for training.

Li et al. (2016) developed a framework for a user simulator and released a research proof of concept which was applied to the movie booking domain. The released proof-of-concept, TC-Bot, was written in Python 2.7 and hard-coded to

support the movie booking domain. Currently, there is no open source and modern user simulator tool for task-completion dialog research. This thesis aims to adapt their framework to the restaurant domain, write it in Python 3.6.0 and apply good software engineering principles with the aspiration that Socrates Simulator can be used for other domains by the dialog research community.

Finally, Facebook recently released the beta version ParlAI. ParlAI aims to provide a standardized and unified framework for developing dialog models. Theyve released a broad set of tools to support training and development of dialog systems for the following domain areas: question answering, goal oriented dialog, chit-chat dialog, visual qa/dialog and sentence completion. ParlAI offers a simplified set of API calls to common dialog datasets (e.g. SQuAD, bAbI tasks, MCTest, etc) and provides a set of hooks to Amazons Mechanical Turk to test ones dialog model again real human testers. While, ParlAI offer an expansive set of tools and datasets, missing from its framework is a user simulator.

1.2. Project Goals

...

Chapter 2: Design

2.1. Introduction

The goal of this thesis is to develop a modularized and production grade user simulator that can be re-targeted for new domains and provide a framework to produce training data and train dialog agents. The overall design for the simulator was inspired by the work done by Li et al. (2016) and the formalization of hidden user agenda models by Schatzmann & Young (2009).

Underlying the simulator is a framework that consists the following components:

- **User Simulator**: An agenda based user modeling component that generates natural language speech utterances to simulate what an actual human would say in the context of task-completion dialog activity.
- **Dialog Agent API**: A set of methods to allow a researcher to provide an agent(s) that simulate how the system / dialog agent would respond
- **Dialog Manager**: A coordinator component that tracks the current state of the dialog and facilitates the conversation between the user simulator and dialog agent/system. Add the end of the simulated conversation, the manager will evaluate and score the conversion.

2.2. User Simulator

The user simulator is responsible for imitating a real user and generating realistic speech utterances. Here we assume the user is an actor that is attempting complete task. For example, the user may want to travel to Japan and is attempting to book a flight there. That user could then interact with a travel agent chatbot in order to get assistance in identifying the appropriate flight and purchasing tickets. In order to model and represent a user, we will utilize the formalization of the hidden user agenda described in Schatzmann & Young (2009).

One of the primary assumptions here is that user has intentionally engaged with the dialog agent in order to complete their task. At the outset of the conversation, the user will have some specific goal in mind (order Indian food, book a flight to Japan, etc). The dialog agent will attempt to learn user's goal by asking the user a set of clarifying questions. Schatzmann and Young introduces the idea of a hidden user agenda as a mechanism to represent the sequence of dialog acts and utterances a user will say in the context of that conversation. At each step of a task-completion dialog, the user is either responding to the dialog agent or initiating a new conversation direction. The user agenda provides an efficient way and formal structure to represent the pending set of dialog acts the user will communicate to the dialog agent.

Socrates Simulator implements the Schatzmann and Young's concept of user agenda and user goal as first class objects. The conversion from the formal representation to code is rather straightforward as there are analogous data structures. We will defer discussing implementation details until the next chapter. The subsections below will further describe how the user goal and agenda is defined.

2.2.1 User Goals

The user goal captures explicitly both user's preferences and missing information needs they are trying to fill. For example, take a user who wants to find an Indian restaurant in Central square for dinner. We can decompose this goal into two distinct components. The first is the user's explicit preferences. In this example, their preferred cuisine is Indian. The second component is implicit and unknown to the user. They are looking for a restaurant or more specifically the name and presumably the restaurant's phone number and address. This information is unknown but can be broken down into discrete pieces of information the user will attempt to elicit from the dialog agent as a request for more information.

Formally, Schatzmann and Young defines the user goal G as $G = (C,R)$, where C consists of constraints or the user's explicit preferences and R represents the user's requests. The constraints and requests are explicitly represented as slot-value pairs. 2.2 below shows how one could represent the goal of user looking for a bar.

$$C = \begin{bmatrix} \text{type} = \text{bar} \\ \text{drinks} = \text{beer} \\ \text{area} = \text{central} \end{bmatrix} \quad R = \begin{bmatrix} \text{name} = \\ \text{addr} = \\ \text{phone} = \end{bmatrix}.$$

Figure 2.1: Example User goal. User wants the name, address and phone number of a cheap bar in central.

The concept of a goal abstractly turns out to be useful in also driving the dialog manager's simulations. We abstract the idea of goal and make it available to both the user and dialog agent api as way to track the internal state of each speaker.

2.2.2 User Agenda

Schatzmann and Young define the user agenda as a [stack] structure of pending dialogue acts [which] serve as a convenient mechanisms for encoding the dialogue history and users state of mind Schatzmann & Young (2009). Formally, at any time t, the user is in a state su and takes an action au,which transitions into an intermediate state su. During this intermediate state, the user will receive an action from the system (machine) am, which will transition dialog to next state su and the cycle will reset. The result is a sequence of alternating turns between the user and system (i.e. su -¿ au -¿ su -¿ am -¿ su -¿), which represents the conversation state over time t.

The user agenda A is stack-like structure which contains all pending user actions. User actions are actualized through popping the stack and the agenda is updated by pushing back onto the stack. A user act is a representation of the users intent, which will eventually be translated into a speech utterance. The stack may also contain other actions that will affect the user when popped. For example, the system can communicate a restaurant suggestion, which would fill the one of the request slots for restaurant name.

At the start of the dialog a new goal is randomly generated from the provided dialog domain. An accompanying agenda is then generated to represent the potential sequential of events.

Figure 2.2: Example User Agenda

Below is an example of the a sample user agenda that Schatzmann and Young provide in the context of a user asking the dialog system for a bar recommendation [6]. The states of the conversation are indexed by time t. Note, Schatzmann and Young use constraints C, which would be the equivalent of inform slots in our representation.

In the first turn, the user simulator generates a set of constraints (bar serving beer in central) and goals (name, address, and phone for a bar that meets the constraints in C0). This set of inform and request slots are translated into a user action stored in A0. When the system initiates the conversation, the user simulator pops two inform actions which translate into the user utterance Im looking for a nice bar serving beer. When the system at t=1, responds Ok, a wine bar. What price range? the agenda updated to include a new inform intent (inform(prange=cheap)). Also added is a negate action, as the user asked beer and not wine.

Over the course of the conversation the agenda is updated, as are the request slots. The conversation ends at t=5, when bye() is popped and the agenda stack is empty. The conversation will then be evaluated based on how well the request slots were filled.

2.3. Dialog Model

2.3.1 Overview

Add paragraph here about high level design and how all component fit together.

Give example of simple conversation flow.

2.3.2 Design Philosophy

At a high level, our aim is to develop a framework to allow dialog researchers to simulate a conversation between the user simulator and dialog agent. The framework needs to be modular so that it can be quickly adapted to support new dialog domains and experiments. Each major component of the framework is represented as a python class. Additionally, I promoted dialog actions, goals, and domain knowledge bases to

first class objects, rather than implementing them at the lower dictionary level. As first class objects, I can standardize the communication, APIs, and management of these pieces and ensure more consistent behavior.

Additionally, we want to provide as much flexibility and freedom to the researcher and limit what is hard-coded. I follow the configuration first approach leveraged in the development of AllenNLP, a deep learning for NLP tool developed at the Allen Institute for Artificial Intelligence. The code is generalized and the user defines particular implementation details in configuration file.

To support a configuration driven approach, each configurable module will have a use defined yaml or json file. For smaller configuration details, the user will prefer using yaml which is more human readable.

The yaml format is simple and has a very low learning curve. It follows a basic key value pair paradigm, where keys have clear semantic meaning and values can be represented in a variety of data structures.

Figure 2.3: Example yaml section.

```
# Dialog Simulation settings
simulation_rounds: 10
max_turns: 8
first_speaker: usersim
reward: 10
simulation_output_path: data/simulated_dialogs/
save_history: True
save_location: "data/simulated_dialogs/"
save_type: json
```

2.4. Simulation Framework Components

All components are represented as python classes. The ensuing section further describe the component's function, design, and any salient implementation details.

2.4.1 Dialog Action

The fundamental unit of transaction for the simulator is the dialog action. It is an explicit semantic representation of the speech utterance that is machine readable. The dialog action consist of three key properties: the dialog act, a set of explicit params, and corresponding unparsed speech utterance.

2.4.2 Goals

2.4.3 Dialog Status

2.4.4 Speaker

The speaker represents an actor that has the ability to speak and comprehend speech utterances. In our framework, the both user simulator and the dialog agent are represented by the same base speaker class. Both actors conceptually are identical in terms of functional behavior, in that they both listen and comprehend speech utterances and in turn respond by speaking. As such, both the user simulator and dialog agent can be represented in the same way to the dialog manager. In fact, the entire conversation round can be expressed in two lines:

Figure 2.4: Psuedo code for conversation round.

```
# Assuming user speaks first
# 1. User Simulator takes turn and speaks
user_action = user_simulator.next(previous_agent_action, turn_number)
# 2. Agent takes turn and responds to user
agent_action = agent.next(previous_user_action, turn_number)
```

The speaker class has four basic functions (*next, reset, get utterance, and parse utterance*) and three properties (*nlg model, nlu model, and dialog status*). When the speaker is initialized, the natural language object and the natural language under-

standing object are passed to the constructor. We abstract away the implementation of how the speaker speaks and parses speech in order maintain separation of concerns and also empower the researcher to be able to experiment with multiple techniques.

The *next* method is the primary driver how the speaker behaves. For the dialog agent class, the next method functions as an API to the simulator. It is assumed that the dialog agent will live and operate external the simulator. The researcher can define how the dialog agent will interact with the user simulator here. For the user simulator, the bulk of the logic will reside here. The *get utterance* and *parse utterances* methods are simply wrappers for the speaker's nlg and nlg objects. The primary parameter for next is the previous dialog action.

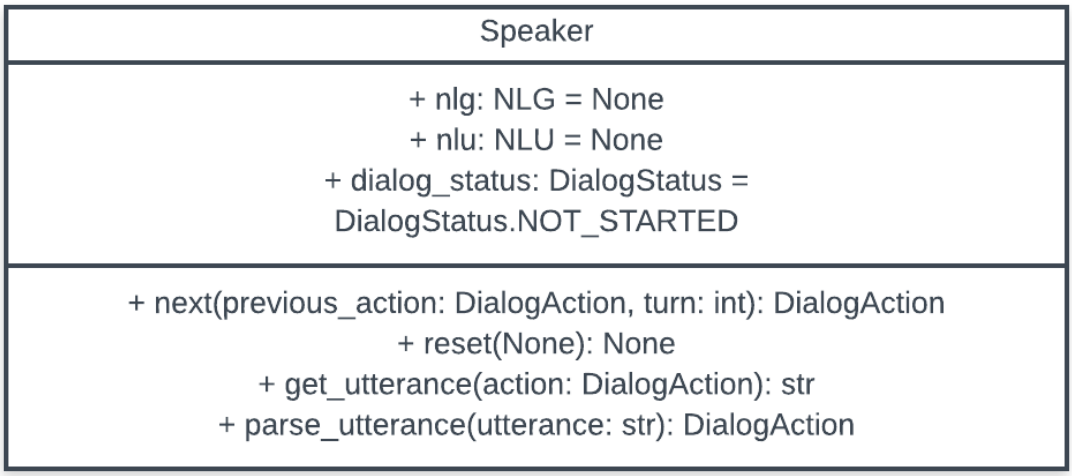

Figure 2.5: Definition of the Speaker class and methods.

2.4.5 Dialog Domain

The domain class standardizes the collection and storage of information about the

Domain Knowledge Base.

2.4.6 Natural Language Understanding

2.4.7 Natural Language Generation

References

Bordes, A. & Weston, J. (2016). Learning end-to-end goal-oriented dialog. *CoRR*, abs/1605.07683.

Li, X., Lipton, Z. C., Dhingra, B., Li, L., Gao, J., & Chen, Y. (2016). A user simulator for task-completion dialogues. *CoRR*, abs/1612.05688.

Rey, J. D. (2017). Alexa and google assistant have a problem: People aren't sticking with voice apps they try.

Schatzmann, J., Weilhammer, K., Stuttle, M. N., & Young, S. J. (2006). A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Eng. Review*, 21, 97–126.

Schatzmann, J. & Young, S. J. (2009). The hidden agenda user simulation model. *IEEE Transactions on Audio, Speech, and Language Processing*, 17, 733–747.