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

We aim to develop a dialog simulation framework, Socrates Sim, to support task completion dialog research. The goal of the framework is to provide a set of tools that will simulate the conversation between a user simulator and a dialog agent in order to generate large training data. Traditionally, dialog researchers would use human testers and annotators to generate quality labeled training data. However, this approach is very expensive and time consuming. The hope is that this framework will help researcher augment human generated test data with synthetic data, which would be generated from simulations with a user simulator.

The Socrates Sim framework will provide a set of tools to define dialog domain, set up a user simulator, and run multiple simulations with a provided dialog agent. To demonstrate the flexibility and robustness of the framework to adapt to new domains, we will implement end-to-end simulations for the restaurant recommendation and move booking use case. The framework will be implemented in Python and made available on github at https://github.com/dhairyadalal/socrates.

Contents

Chapter 1: Development

This chapter discusses the tools and methodologies employed in the code development of this system.

1.1. Development Language

The framework was written in Python 3.6, which at the time of submission is the latest supported python release. It is worth noting that the research implementation of the user simulator described in Li et al. (2016) was written in Python 2. Python 3 is the preferred version for production python products. Most major data science and machine learning research python libraries no longer support Python 2. Python 3 provides many useful features and performance upgrade that make writing and deploying python projects more efficient and effective. In addition to updated syntax that allows for more expressive coding, the standard library was extended to support new data types (ordered dictionaries, enumerated types, data classes). Additionally, Python 3.5 introduced type annotation, which allows for the writing of cleaner, better documented, and unambiguous code. We describe type annotations further below.

The framework was written to adhere to PEP8 standard and all method signatures have type annotation. The hope is that good software documentation and standard coding styles will allow future contributers to easily debug, modify, and

build new modules for the framework.

1.2. Development Tools

The framework was developed using the PyCharm IDE. Pycharm was selected for its ease of use for rapid development, python debugging tools, and integration with Github. Various python libraries were used in the development of the framework. The following third party python packages had significant impact on the development of the framework:

- yaml: The yaml library provides a set of function to read and write yaml files. It was primarily for the loading and saving of the various configuration files.
- json: The json library provides a set of functions to read and write json files. It was primarily used for the loading and saving of various configuration files and the serialization of the simulated dialogs.
- spacy and nltk: Spacy and NLTK are natural language processing libraries that provides standard NLP tools like part of speech tagging, dependency parsing and, named entity recognition. They were used primarily for developing the base natural language understanding and natural language generation models.
- pandas: The pandas library provides robust, efficient, and flexible data structures that can be used for data science. The pandas dataframe was used to represent the in memory knowledge base and execute various logical queries.
- opennmt: OpenNMT is python library used for neural machine translation. OpenNMT was used the training of the NLU and NLG models.
- memory-profiler: This library was used to measure the memory consumption of the framework.

The code base is stored on Github.com and uses git for version control and bug tracking. Github is an online, cloud based, software platform used for sharing and hosting code bases. Github provides various collaboration and version tracking. It is one of the most popular platforms for code sharing and collaboration.

The Sphinx tool was used for documentation generation. Sphinx automatically extracts method docstrings and generates code documentation as rendered makrk-down files. For hosting, I used the open source and free Read the Docs service, which hosts code documentation websites.

Chapter 2: Results

This chapter discusses the performance tests and results of the Socrates Simulator.

2.1. Experiment Overview and Setup

Socrates Simulator is novel as a open source end-to-end dialog simulation framework. As a result, it was challenging to develop a meaningful benchmark to evaluate against. We ultimately chose the TC-Bot framework, which is the research implementation of end-to-end neural dialog framework described in Li et al. (2017). We measured two aspects of the Socrates simulator, its scalability and memory consumption over the course of running multiple simulations. The goal was to show that Socrates Sim is generally usable and overall performance does not significantly degrade with expanded usage across different domains and increased simulation runs.

TC-Bot was the academic inspiration for this project, and share similar objectives in training and evaluating task completion dialog agents. However, there are several key distinctions between TC-Bot and Socrates Sim. First, the neural architecture for training the dialog agent is tightly coupled with the overall framework for TC-Bot. Second, TC-Bot was hard coded for a limited movie booking use case. Finally, the training data used by TC-Bot was collected with Amazon Turk and not publicly released. They did provide some of the intermediate representations (Python

2.7 encoded pickle files) of the data (e.g., dialog acts, slot values, etc). However, these intermediate representation were particularly optimized for their specific use case and were unusable general usage. Adapting TC-Bot to a new domain would have required non-trivial work rewriting the neural framework, which is outside of the scope of this project. To test TC-Bot, we ran framework as is, with no changes to the domain or underlying framework.

Initially, performance testing was conducted a MacBook Pro laptop with a core i7 processor and 16 GB of memory. However, TC-Bot's performance was significantly reduced due to its neural architecture which performed best on a gpu (graphical processing unit) enabled machine. In order to ensure a fair comparison, all performance tests were run on a gpu enabled server. Specifically the server was running Ubuntu 14.04, with a core i9 3.3 ghz 10 core cpu and 64 gb of RAM. Additionally, the server had two gpus, a Titan Pascal and a Titan XP 12 gb GPU.

A master script was written to coordinate iteratively calling each framework with a variable simulation count parameter. The framework would be called as an isolated subprocess of the master scrip to ensure more accurate measurements.

2.2. Runtime Performance Experiment and Results

The goal of these tests was to measure how effectively the framework's runtime scaled with increased simulation rounds. Runtime is defined here as the elapsed time taken to run n simulations, where is n is a positive integer. Starting with 1 simulation, we iteratively increased the simulations by 500, until we ran a total of 50,000 simulated dialogs. We did not capture the write time to save the simulated dialogs to disk. This is usually a linear cost proportional to size of the stored dialogs in memory.

In total, we ran 6 different performances tests. Four tests were run on the Socrates Sim framework and two on the TC-Bot framework.

For the first two tests we compared the performance of end-to-end dialog simulations using a rules based approach. For both frameworks, we used the movie booking use case. In the Socrates Simulator, both the dialog agent and user simulator are rules based speakers, and have rule based nlu and nlg modules. In TC-Bot, the user simulator is rules based and the dialog agent is DQNN neural model, that is being trained on a reward signal after each simulation round. Both frameworks has essentially a *O(N)* runtime, as the simulation size grew, as you can see in Figure 2.1. However, the Socrates Sim performed overall more efficiently, taking on average about 1 minute and 51 seconds to run 50,000 simulations. In contrast, TC-Bot took on average 10 minutes and 44 seconds. A large parts of this can be attributed to the fact that TC-Bot tightly couples training the dialog agent with the dialog simulation. That is, after each round, TC-Bot runs stochastic gradient descent on the reward function in order to inform the dialog agent's action in the proceeding round.

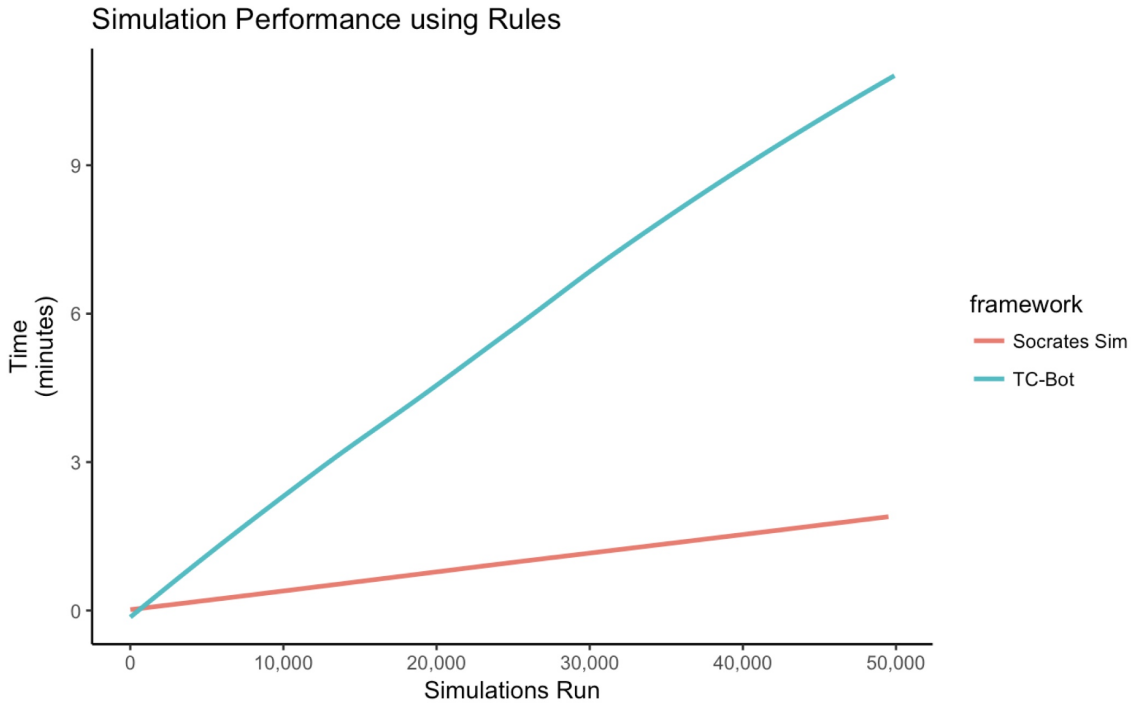

Figure 2.1: Runtime of Socrates Sim vs TC-Bot with rule based user simulator

The next two tests, we compared the performances of both frameworks using model based actors. As with the first two experiments, we used the movie booking use case. For Socrates Sim, we used OpenNMT trained nlu and nlg module for the user simulator and dialog agent. For TC-Bot, both the use simulator and dialog agent were DQNN agents. They were both trained for 100 epochs over 500 simulations, with a batch size of 16. Additionally, the experience replay pool was set to 1000 and the dqn hidden size was 80. TC-Bot took about 5 hours to train both the user simulator and dialog agent end-to-end.

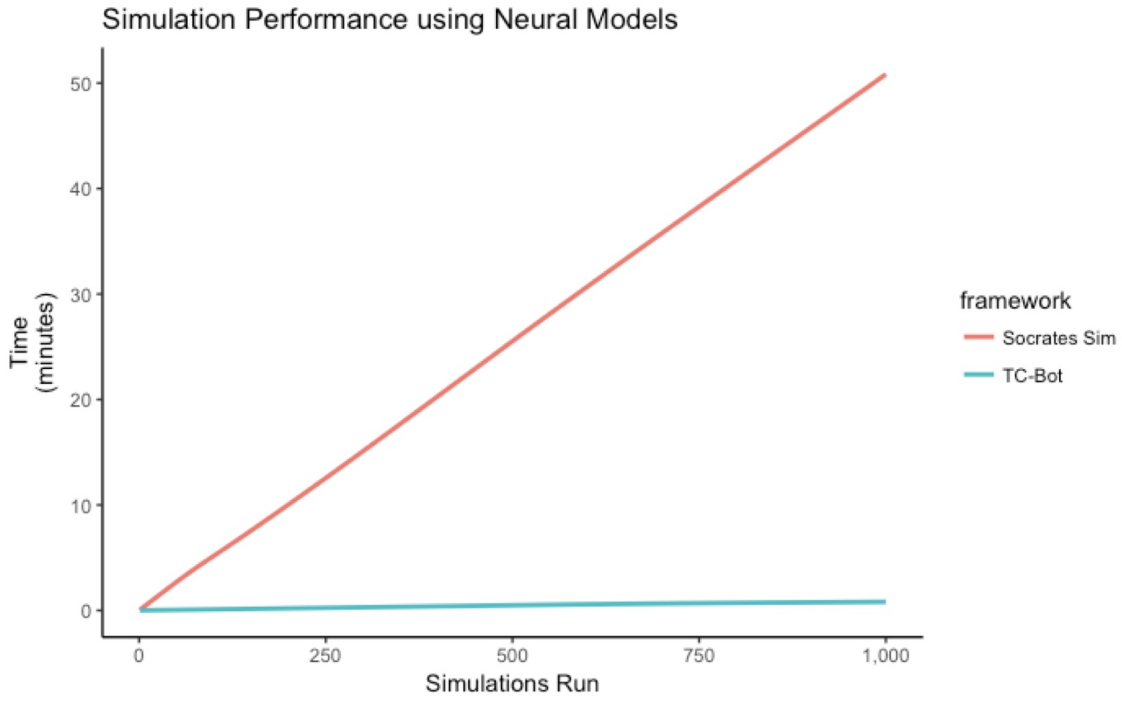

Figure 2.2: Runtime of Socrates Sim vs TC-Bot with model based user simulator

In Figure 2.2. we see that Socrates Sim takes about 50 minute to 1,000 simulations. Socrates Sim continues to have linear growth, though the runtime line is much steeper. This can can be attributed the in-optimal deployment of the OpenNMT model for both the nlg and nlu. Since OpenNMT is currently a command line tool, it expects its inputs as text file and makes batch predictions. The OpenNMT tool had

to be rigged to support on demand predictions. We to write the inputs to a text file and invoke OpenNMT command line as subprocess in order to generate on-demand predictions. Each invocation carries the overhead of loading the model into memory, preprocessing the input, and running the input through NMT network. As we see below, this is quite expensive. A preloaded model in memory would be more effective. As the goal of this project is the demonstration of the end-to-end framework, additional time was not spent to custom build an optimized NMT model. In contrast, we find that TC-Bot does perform much more efficiently. While its growth is still linear, it can run 1,000 simulations in about 30 seconds. Once trained, TC-Bot is rather efficient at running simulations.

Finally, our set of experiments involve testing runtime performance across domains. As TC-Bot is hard-coded for just movie booking use case, it was not included in our performance tests.

2.3. Memory Consumption Experiment and Results

References

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

Chewning, C., Lord, C. J., & Yarvis, M. D. (2015). Automatic question detection in publication classification natural language.

Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M. E., Schmitz, M., & Zettlemoyer, L. (2018). Allennlp: A deep semantic natural language processing platform. *CoRR*, abs/1803.07640.

Li, X., Chen, Y., Li, L., & Gao, J. (2017). End-to-end task-completion neural dialogue systems. *CoRR*, abs/1703.01008.

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.