Prompt Selection with Reinforcement Learning in an AT&T Call Routing Application

Charles Lewis
Giuseppe Di Fabbrizio

AT&T Labs – Research
180 Park Ave. – Florham Park, NJ 07932 – USA
{clewis, pino}@research.att.com

10.21437/Interspeech.2006-490

Abstract

Reinforcement Learning (RL) algorithms provide a type of unsupervised learning that is especially well suited for the challenges of spoken dialogue systems (SDS) design. SDS are constantly subjected to new environments in the form of new groups of users, and RL provides an approach for automated learning that can adapt to new environments without costly supervision. In this paper, we describe some results from experiments with RL to select prompts for a call routing application. A simulation of the dialogue outcomes were used to experiment with different scenarios and demonstrate how RL can make a system more robust without supervision or developer intervention.

Index Terms: spoken dialogue systems, reinforcement learning, call routing

1. Introduction

Reinforcement Learning (RL) systems simultaneously learn and perform without supervision. Instead of learning a single, static solution, RL implementations learn and adapt continuously over time. This makes RL techniques applicable to a variety of real-world problems that defy supervised machine learning (ML) solutions.

Many environmental factors (such as the volume of calls received, the hours that the application is in operation, and the geographical region where the application is deployed) can affect the operation of an SDS application, yet are unknown at the time that the application is designed. Based on their expertise, User Experience (UE) engineers have to make many non-trivial decisions, such as the system's semantic scope (e.g., call-types in the case of call routing systems), the dialog manager strategy which will drive the human-machine interaction, and many other facets of these applications. Often, these experts cannot agree on the best type of opening prompt, for example, or the best compromise between hand-holding for and empowerment of the user.

After the application is deployed and its performance analyzed, the UE expert can use this data to adjust the application and deploy it a second time. This is repeated as often as necessary. Each re-deployment requires human intervention for data transcription, data analysis, re-authoring, and quality assurance beforehand, engineering resources for the re-deployment itself, and monitoring after to determine the effects of the re-deployment and to determine if further tweaking is necessary.

The goal of this experiment was to explore the potential for an RL solution to reduce the need for re-deployment by assuming responsibility for prompt selection. When the prompts for a system are created, the designer has a number of ways of phrasing each one. For example, the system may take the initiative and tell the user what their options are (closed or system initiative prompt), or the system may invite the user to express their needs in their own words (open or user initiative prompt). Some prompts use a fictional personality, and some prompts may try to reassure the user with friendly instructions. The effectiveness of each kind of prompt depends on the situation and how well they are received by the user.

The unsupervised learning of an RL system can have great benefits here, where human supervision of changes to the system is costly and time-consuming. Rather than force the UE expert to decide between prompts, this approach defers that decision and provides a method to let the system exploit the optimal prompts over time. Potentially, this can make dialogue designs more robust by allowing the decision to be made at run-time, based on feedback from the environment.

2. The Application

We applied this approach to a call routing application used by AT&T's small business customers to report and track service problems. This application tries to elicit enough information from the caller to route the call to a call center that specializes in their request.

Call centers are sensitive to call volume, so the application tries to route calls as specifically as possible. By accomplishing initial request type identification, the application can reduce the amount of time that human operators spend speaking to customers, and increase the number of customers that the call center can handle. Conversely, the more unclassified calls received by a call center, the fewer customers that they can handle.

The purpose of the call routing process is to elicit the user's intent. If the caller asks to be routed directly to a human customer service representative (CSR), a special prompt is used to try to convince the user to interact with the automated system. In this application, the UE expert created four potential prompts for this situation, which we will refer to as CSR prompts. They run from completely open to completely closed, and utilize different levels of hand-holding and reassurance. In our

experiments, the choice of which prompt to use was made with a RL algorithm.

3. RL, and RL for Voice Applications

RL-type techniques have been used in voice applications before. In ([3] and [7]), RL systems are used to plan high-level dialogue strategies. These systems evaluated a design problem and created an optimal solution for deployment. This application of RL does not derive a single, optimal solution, rather, it implements an adaptive one. The approach used here will demonstrate a way for the UE expert to author a range of possible prompts and then leave it to the system to determine which one should be used.

At the completion of a task (or at some intermediate step in a task), the RL system receives feedback which it uses to refine its behavior in future episodes. Unlike supervised learning, which requires a large set of labeled data, RL can be put into motion without any pre-compiled model of its environment. There are many variations of the RL problem and possible solution implementations (as described in [2], [4], and [8]).

The components that most RL systems have in common are a *policy* to guide its decisions, a *value function* to describe the value that the system puts on a state (or a state/action pair) within the course of problem-solving, and a *reward function* to describe the environment's reinforcement of a course of action.

The policy is the algorithm that makes the decisions, and the data structures that support it. We denote the current policy as π, and the optimal policy as π^*. π is the subject of continuous refinement in RL, with the goal, of course, of achieving π^* or close to it.

The value function describes the value that the system puts on a particular state, or state/action pair. This function relies on the experience of the system to ascribe a long-term utility to available actions, and informs the choices made by the policy. Value functions as they are used in this paper are based on the action taken in a particular state. These are called the *action-value* functions in the literature and denoted as *Q(s,a)*. We will refer to these too as simply value functions. The most accurate action-value function, *Q*(s,a)*, will result in the best policy, π^*.

The reward function describes the feedback from the environment, it is what the value functions try to predict. Part of the RL problem can be thought of as a refinement of the value functions to more closely approximate the reward function. It is possible for there to be both intermediate rewards and final rewards in the decision making process. Only final rewards will be provided in this application.

Learning occurs with these components by a process called Iterative Policy Generation (IPG). In this process, there is a feedback loop between the policy, π, the value function, Q(s,a), and the reward function:

1. The system makes a decision using policy π.
2. The reward function provides feedback on the decision.
3. Decision value function is refined
4. A new policy, πl, is created, based on the new value function. This is the policy used in the next dialogue

Because the new policy is based on an improved value function, it will, on average, achieve improved rewards. As the number of iterations increases, and the value function improves, the policy approaches π^*. In order for this system to work, the policy must take advantage of the accuracy of the value functions to maximize the expected reward. How this is done depends on the RL implementation used.

The so-called Monte Carlo approach relies entirely on experience with the environment to improve policy. The two critical operations are refining the value functions (performed between well-defined episodes), and deciding which action to take from a given state s (performed during episodes). When the system receives reinforcement from an episode, it integrates this new data into relevant value functions. If the reward function never changes, a straightforward average of all returns from a state-action pair will provide a continuously improving approximation of the true value of the pair as more data becomes available, by the law of large numbers. In this scenario, there is no need to weigh new data any differently from old data. If we cannot make this assumption, however, we must discount the old values as new data becomes available. This is done by scaling the effect of new rewards with a step-size (α) as follows:

$$Q(s,a)' \leftarrow Q(s,a) + \alpha (r - Q(s,a)) \qquad (1)$$

This distinguishes RL from systems that use statistical optimization without consideration for the order in which the data was received. This makes an RL system more effective for a dynamic environment where typical optimization is not a perfect fit.

4. Data Collection

The application, including the application logic and the prompts used, was designed by an AT&T User Experience expert. This design was then implemented using Florence, the dialogue manager in AT&T's VoiceTone® system [1]. The application was deployed under the VoiceTone® platform, which provided the logging used for data collection. RL was not applied in the data collection. During collection, the system selected a prompt randomly at each decision point.

Over the three months that the application was in use, data was collected on 9,786 dialogues. Of these calls, 845 used a CSR prompt. The data collected on each call included the dialogue states entered, prompts used, number of re-prompts due to silence timeouts or speech recognition rejections, and the routing destination of the call. We simulated dialogue outcomes in the experimental phase based on the frequency of each outcome for each CSR prompt in the collected data.

The general idea of the RL reward is to give successful dialogues a higher score than non-successful ones. The question of what constitutes a "successful" dialogue has been examined before, for example in the context of predicting problematic dialogues [9]. In our case, we are not limited to simply successful or not-successful: we can also assign scores that correspond to varying levels of success after the dialogue is complete.

The most obvious criterion to use is how the routing was handled. If the call was routed to a call center it was successful to some degree. If the user hung up before the call could be routed, it was not. The reason for the hang-up is unknown to us, and could be unrelated to the system design, but for our purposes this type of dialogue was considered not-successful.

There is one routing destination, the general CSR line, that is used as a fallback when the system cannot gather enough information to choose one of the others. This occurs when the system has given up on eliciting more specific information from the caller. Although this is not as bad as a complete disconnect, we would prefer a well-specified routing. This outcome was assigned a score greater than the score for a hang-up, but less than the score for more specifically routed calls.

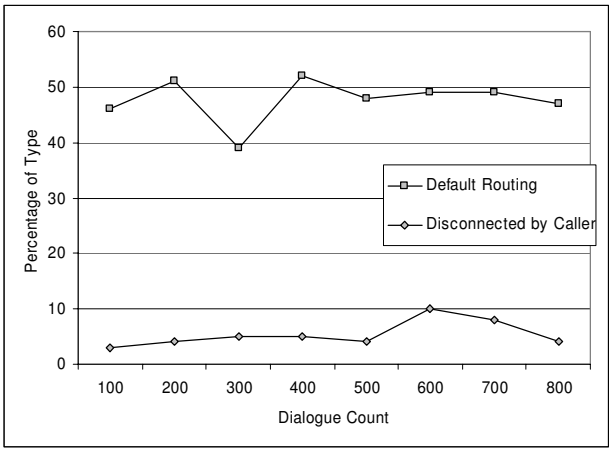

Figure 1 *Number of non-successful dialogues from gathered data.*

After a call was scored, the reward value was used to update the appropriate value function, as per Equation 1.

For comparison to the experimental data, Figure 1 shows the number of calls disconnected by the user before routing and the number of calls that were routed to the default CSR line (default routing) during data collection with random prompt selection. The graphs in this paper show one data point for every 100 dialogues that used a CSR prompt. This value is presented as a percentage of calls of each type.

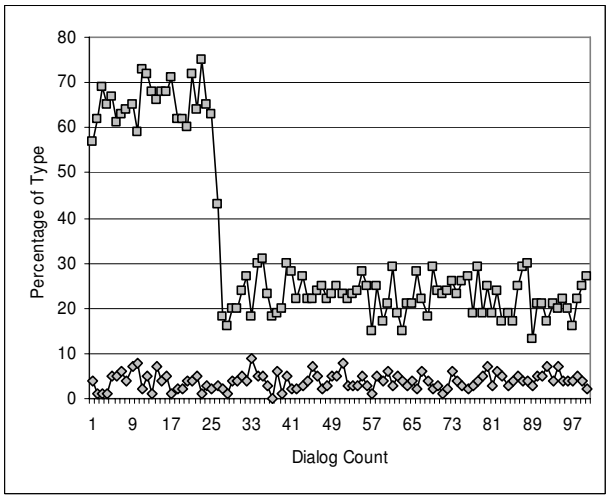

Figure 2 *Number of non-successful dialogues in a simulation of a stable environment. (Legend as per fig. 1)*

5. Simulated RL in a Stable Environment

Figure 2 is a typical outcome of the first set of experiments with this simulation, where a data point was computed every 100 dialogues, for a percentage of each type. The number of dialogues with default routing drops sharply after around 2500 dialogues. The number of dialogues where the caller hangs up remains fairly steady, despite significant differences in the hang-up rates between the prompts. Because many more calls result in default routing than in the caller hanging up, prompt selection is dominated by consideration for the former.

In the calls leading up to the steep drop in default routing, the system develops the value function for each prompt. The drop occurs when the prompt that results in the highest rewards has attained a value high enough to keep it in the lead. Fluctuation of value scores is illustrated in more detail in the next section.

The results of this simulation demonstrate the potential for the Monte Carlo approach to improve the routing of calls. In a real deployment, over time, the system would have developed a preference for the prompts that were more successful at eliciting information from the user, resulting in fewer calls routed to the default operator.

6. Simulated RL in a Dynamic Environment

The environment in this simulation is produced by the dialogue outcome models, which are compiled from the collected data. To simulate a change in the environment, the models were changed halfway through each run. At this point, the outcome model for the top performing prompt is switched with the model for the worst performing one. Although this isn't a realistic change in the environment (the simulated user reactions to the prompts), it is drastic enough to demonstrate the adaptive effects of the RL algorithm. As shown in Figure 3, this had an immediate affect on the simulated percentage of default routing. Immediately after the outcome models were switched, the number of calls with default routing returned to its initial level, before RL took effect. Within 100 dialogues, the percentage of default routing dropped again but much more quickly than it did at the beginning of the process. This rapid re-adjustment lowered the number of default routing calls down almost to the level seen before the model change. This pattern was typical of the runs in this experiment.

In Figure 4, we can see what happened behind the scenes in one of these runs. This chart shows the value for each of the prompts as new data is received. In the first half of the run, the value for prompt 031 gradually increases to become the highest, and then the number of calls routed to the default destination (Figure 3) declines drastically. This is the same behavior that we saw in the previous experiment, without the model change.

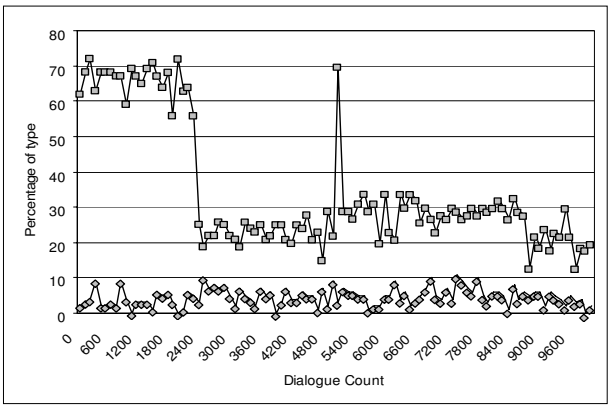

Figure 3 *Number of non-successful dialogues in a dynamic environment. (Legend as per fig. 1)*

Halfway through the run, the model that generated the results for prompt 031 dialogues, the most valued prompt, was switched with the model that generated results for prompt 032, the least valued prompt. This is where Figure 3 shows a large up-tick in the number of exceeded re-prompt calls, and where the value for prompt 031 drops precipitously in Figure 4.

Within a hundred dialogues of the up-tick, however, the number of calls routed to the default destination is down to a much lower level. This adjustment happened very quickly because the second-most valued prompt, prompt 030, takes over as soon as the value of prompt 031 declines. This immediately lowers the percentage of default routings. Eventually the model for prompt 032 (originally the model for prompt 031) takes the lead again, and the percentage of calls routed to the default call center returns to the same level seen before the model switch.

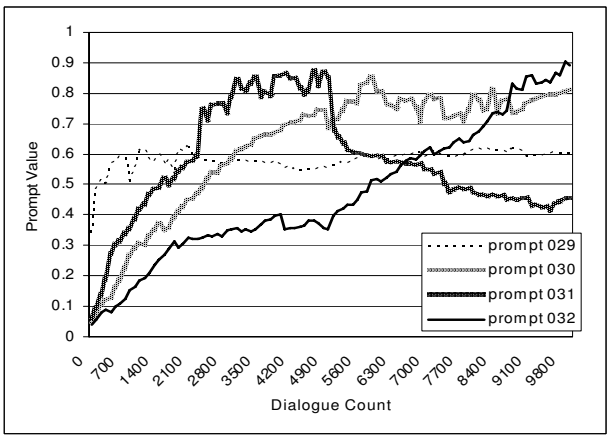

Figure 4 *Prompt scores in a dynamic environment.*

7. Conclusions

This brings to light an aspect of exploration that is not typically mentioned: beyond its use for the initial exploration to arrive at the best action, beyond using exploration to make sure that the most highly-valued action is still the most highly-rewarded, exploration in RL also maintains the accuracy of the value functions of all of the state/action pairs throughout the lifetime of the application. In this experiment, when the top-performing action stopped performing, the second-best action quickly took its place, limiting the effect of the environmental change. This controlled degradation of performance provided a safety net for the system until it was able to readjust the values of the affected prompts.

These experiments demonstrate that it is possible to frame spoken dialogue system prompt selection as an RL problem, and that it is possible for the Monte Carlo RL technique to provide continuous, unsupervised learning for this task. In the first set of experiments, it was demonstrated how this approach works in a new environment, with no assumptions made about the relative values of each action. In the second set of experiments, the environment was dynamic and the system was tested to see how well it could adapt. This provided an interesting demonstration of how the exploration inherent in RL systems makes them more robust to change.

8. References

[1] G. Di Fabbrizio and C. Lewis, "Florence: A Dialogue Manager Framework for Spoken Dialogue Systems," ICSLP 2004, 8th International Conference on Spoken Language Processing, Jeju, Jeju Island, Korea, October 4-8, 2004.

[2] L.P. Kaelbling, M.L. Littman, "Reinforcement Learning: A Survey," Journal of Artificial Intelligence Research 4, May 1996, 237-285

[3] E. Levin, R. Pieraccini, and W. Eckert, "A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies," IEEE Transactions on Speech and Audio Processing, Vol. 8 No. 1, January 2000

[4] D. Litman, M. Walker, and M. Kearns, "Automatic Detection of Poor Speech Recognition at the Dialogue Level," ACL-1999

[5] Mitchell, T., Machine Learning, McGraw-Hill, 1997.

[6] S. Singh, M. Kearns, D. Litman, and M. Walker, "Reinforcement Learning for Spoken Dialogue Systems," NIPS-1999

[7] S. Singh, D. Litman, M. Kearns, and M. Walker, "Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System," Journal of Artificial Intelligence Research 16 (2002), 105-133

[8] Sutton, R. S. and Barto, A. G., Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998

[9] M. Walker, I. Langkilde, J. Wright, A. Gorin, and D. Litman, "Learning to Predict Problematic Situations in a Spoken Dialogue System: Experiments with How May I Help You?" In Proc. 1st Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), 2000.