Using Reinforcement Learning for Dialogue Management Policies: Towards Understanding MDP Violations and Convergence

Peter A. Heeman, Jordan Fryer, Rebecca Lunsford, Andrew Rueckert, Ethan O. Selfridge

Center for Spoken Language Understanding
Oregon Health & Science University
20000 NW Walker Rd., Beaverton, OR, 97006

Abstract

Reinforcement learning is becoming a popular tool for building dialogue managers. This paper addresses two issues in using RL. First, we propose two methods for finding MDP violations. Both methods make use of computing Q scores when testing the policy. Second, we investigate how convergence happens. To do this, we use a dialogue task in which the only source of variability is the dialogue policy itself. This allows us to study how and when convergence occurs as training progresses. The work in this paper will help dialogue designers build effective policies and understand how much training is necessary.

1. Introduction

Reinforcement learning (RL) is becoming a popular tool for building the dialogue manager of a spoken dialogue system (SDS) (e.g., [1, 2, 3, 4, 5]). Much is known about RL from a theoretical point of view. We know that if a dialogue task can be formulated as a Markov Decision Process (MDP), RL is guaranteed to converge asymptotically to the optimal policy [6]. However, in the practical applications of RL, especially for building dialogue managers, more work is needed.

One area where more work is needed is in MDP violations, as most dialogue tasks cannot be formulated as an MDP. Although some MDP violations might have a minor effect on RL’s ability to find an optimal policy [6], others can be detrimental [8]. In this paper, we introduce two techniques for finding them. Although these techniques are not guaranteed to find all violations, they serve as a starting point for further exploration.

Another area where more work is needed is in how RL converges. Once RL finds an optimal policy will it stay from it? If so, how long must one wait until RL will not stray from it again? Understanding how convergence happens will help dialogue designers better understand what to expect from using RL to find a dialogue policy. However, studying convergence is difficult, as there are many sources of noise in most SDS applications of RL. This includes minor MDP violations, and noise from the user simulation. In this paper, we use a domain that is simple enough in which we can remove these sources of noise, and investigate how convergence takes place.

The rest of the paper is organized as follows. First, we describe a dialogue management framework that combines RL with the Information State Update (ISU) approach [7]. Second, we describe our dialogue task and its RL-ISU implementation. Third, we present two methods for discovering MDP violations and demonstrate them on our domain. Finally, we analyze how convergence is achieved, using a version of our domain without MDP violations.

2. RL-ISU Framework

We use the framework described in [7] to build spoken dialogue systems (e.g., [8]). This framework combines RL and the Information State Update approach (ISU).

RL learns a dialogue policy by determining what action (i.e., utterance) is best for each possible state (i.e., dialogue context). It does this by estimating a Q score for each state-action pair s,a, which is the cost of getting to the final state from state s using action a, and then following the current policy. RL starts with an initial policy, and runs large numbers of dialogue simulations in which it interleaves estimating the Q scores with refining the policy.

ISU [9] allows us to express the system in terms of a knowledge base (information state) and a set of rules that update this knowledge base. At a minimum, two types of rules are used, action selection rules that determine what the system will say next, and understanding rules, that determine how to incorporate both what the user just said and what the system just said into the system’s knowledge base. The update rules are specified as a set of preconditions and effects. As it can be very difficult to determine all of the preconditions for the action selection rules, RL is used to determine additional preconditions that optimize a cost function. The results of this paper are not specific to the RL-ISU framework, but to any application of RL.

3. Car Buying Domain

For this paper, we use a slot-filling task in a car-buying domain, similar to a number of RL-based dialogue applications (e.g., [1]). The system has 2000 cars in its inventory. The cars have H different attributes, including model, color, year, and mileage. Each car has a different set of values for the H attributes.

The system has thirteen different speech actions it can choose from. Each car has a different set of values for the H attributes. The system has thirteen different speech actions it can choose from. Each car has a different set of values for the H attributes. The system has thirteen different speech actions it can choose from.
the attributes, one for reporting the matching cars, and one for ending the dialogue. The action selection rules have preconditions that ensure that an attribute is asked at most once, and that the system ends the dialogue after it reports the matching cars. The system has two understanding rules: one that records what question the system asked, and one that records the user’s response. We also include a deliberation rule that computes the matching cars after each user response. The system’s information state has variables that indicate the values of the attributes that the user has given it, whether the system has reported the results to the user yet, and which cars match the attributes known so far. We also use the ISU approach for the user simulation.

For RL, we use a reduced set of the ISU variables, namely 11 boolean variables to indicate which attributes we know, the variable that indicates whether results have been reported, and a variable to indicate how many cars match, quantized into 4 buckets: 1, 2-5, 5-15, and 16-2000. Quantizing the number of options available is commonly done when applying RL to dialogue systems, as it keeps the state space at a reasonable size (e.g. [1]). Because each user has one of the 2000 cars in mind, any sequence of queries will result in at least one car matching.

RL requires a cost function. As is typically done, we use the sum of dialogue length and solution quality. Dialogue length is measured as the number of system utterances (including the exit). Solution quality is 0 if exactly one car is reported; otherwise, 5 points for each additional car reported.

Learning a System Policy: For each experiment, we learn a set of policies for some amount of training.

Learning a System Policy: For each experiment, we learn a system policy, and so the domain could have an MDP violation.

From the above, we know that during testing, some of the Q\text{train} and Q\text{test} scores for the state-action pairs of each policy at each test session, as well as the average of the maximum difference for each policy. Both curves are smoothed by averaging over neighboring test sessions (5 before and 5 after).

As can be seen, after an initial improvement during the first 1000 epochs, no further improvement is seen. After 5000 epochs, the average difference is 0.101 while the maximum difference is 1.298. The failure of these curves to tend towards zero strongly suggests that RL is unable to learn the optimal policy, and so the domain could have an MDP violation.

Figure 1: Finding MDP Violations using Q\text{test} Scores

Method 1: Q\text{train} versus Q\text{test}: During training, RL keeps a Q score estimate for each state-action pair s-a. As training progresses, the Q scores should converge to the values for the optimal policy. We refer to these as Q\text{train} scores, as they are determined during training.

As we are learning a policy, we conduct test sessions in which we evaluate the current policy (with no exploration) with the simulated users. During a test session, in addition to computing the average dialogue cost, we track the average cost from each state-action pair to the end of the dialogue. These are similar to the Q\text{train} scores, but are based only on the current test session. We refer to these as Q\text{test} scores.

To show the use of the Q\text{test} scores, we learned 4 policies for 85,000 epochs. Figure 1 reports the average difference between the Q\text{train} and Q\text{test} scores for the state-action pairs of each policy at each test session, as well as the average of the maximum difference for each policy. Both curves are smoothed by averaging over neighboring test sessions (5 before and 5 after).

As can be seen, after an initial improvement during the first 1000 epochs, no further improvement is seen. After 5000 epochs, the average difference is 0.101 while the maximum difference is 1.298. The failure of these curves to tend towards zero strongly suggests that RL is unable to learn the optimal policy, and so the domain could have an MDP violation.

From the above, we know that during testing, some of the Q\text{train} and Q\text{test} scores are not converging and that at least one of the state action pairs differs on average by 1.298. To understand what is happening, we investigated a particular instance in which the maximum difference was 2.219 at 85000 epochs of training. For a particular state-action pair, the Q\text{train} value was 4.181491 while the Q\text{test} value was 6.400. The Q\text{train} value was based on 17931 training instances, so it is not the case that it is a state-action pair that was just recently seen and lacks enough data. The Q\text{test} value is based on 5 instances; however, this is an exact estimate for the current policy as it is based on the exact distribution of users.

Method 2: Parent Q\text{test} Scores: The previous method helps indicate whether an MDP violation occurred, but does not tell us why. MDP violations result from not keeping track of enough of the history of previous actions or states in the current state. In other words, it is from inappropriately merging different histories of state-action sequences into a single state. To assist in finding a violation, we introduce parent Q\text{test} scores.

Say that during testing, we observe that s_i under the policy action \pi(s_i) sometimes transitions to s_k, and that s_j under
the policy action $\pi(s_j)$ also sometimes transitions to $s_k$. Of course, $s_i$-$\pi(s_i)$ and $s_j$-$\pi(s_j)$ can transition to other states as well, which for our car domain, will depend on the number of cars that match. Our $Q_{test}$ scores track what the average cost is to reach the final state for each state-action, including $s_k$-$\pi(s_k)$. In this section, we also propose keeping track of the costs for each path into $s_i$, which in this case is $s_i$-$\pi(s_i)$ and $s_j$-$\pi(s_j)$. Let’s call the cost of going from $s_i$-$\pi(s_i)$ to the end in which the previous state is $s_i$ as $Q_{test}(s_i\pi(s_i)|s_i\pi(s_i))$, the parent $Q_{test}$ score. From the MDP assumptions, one can prove that $Q_{test}((s_i\pi(s_i))|s_i\pi(s_i)) = Q_{test}((s_i\pi(s_i))|s_j\pi(s_j))$. This is to say that the cost from a state-action pair to the end does not depend on which state was previously in. Using the data test sessions, we can empirically look for violations in this equation. In other words, we can look for places where we are inappropriately merging different histories into the same state.

To illustrate this, consider Figure 2, which is based on the example we used for the previous test. For the state CMY5, the current policy action is askCylinders. It has a $Q_{test}$ score of 4.126. Three different states can transition to it, CM5, CM15, CM16. These states differ in terms of the number of cars (2-5, 6-15, 16-2000) that match the new set of attributes. However, we are interested in the transitions to CMY5. In the figure, we also give the parent $Q$-test scores for these 3 transitions. If there is not a MDP violation, the parent $Q$-test scores should equal the $Q$-score. However, this is not the case. The parent $Q_{test}$ score from CM5 is 3.667, while CM15 is 3.775, and CM16 is 4.192. Thus, how we get to CMY5, either through CM5, CM15, CM16 influences the cost to get to the end of the dialogue from CMY5. This is a violation of the MDP assumptions.

![Figure 2: Using parent $Q$-test scores to find MDP violations](image)

Although it does not tell a dialogue designer exactly what the violation is, this method will pinpoint where the violation occurs. For our problem, this allows us to deduce that the problem is that we forget the number of cars that previously matched. If 2-5 cars previously matched, we are much likely to now have only two cars match than if we previously had 16-2000 cars match. This in turn influences how likely the next question askCylinder is to get one matching car, and thus allow the system to report the number of cars without a penalty.

Note that if we find a discrepancy than there is an MDP violation. However, this method can only detect violations that show up in a single policy, not across policies. For example, say that the order that questions are asked affects the $Q$ score but the order is not saved in the state representation. In any one policy, the order might be fixed, and so this method will not find a discrepancy in the parent $Q_{test}$ scores.

In summary, the above two methods can help a dialogue designer determine whether there is an MDP violation, and if so, help determine the cause of it. More work is needed to determine the generality of these methods.

5. How Does Convergence Happen?

We now examine how convergence happens. Following standard practice (e.g., [1, 4]), we examine how the dialogue cost improves as the number of training epochs increases. We then present additional measures to better understand how convergence is happening. We examine how it happens at the policy level, how the $Q$ values of state-action pairs converge, and how convergence relates to exploration of the state-action pairs. Though this is only a case study of one dialogue task, this work should give us a sense of how convergence occurs in general.

Studying convergence, even for a particular domain, can be difficult. First, there might be a set of policies that are all optimal. For example, the order of some of the system actions might not matter. Hence, we focus on achieving a certain dialogue cost, rather than whether the policy changed or not. Second, determining whether a policy achieves a certain cost can be difficult, as there might be noise from the user simulation. We avoid this issue by testing on the static user distribution, as explained in Sec. 3. Third, convergence in RL is only guaranteed if there are no MDP violations. We use our car-buying domain with two buckets rather than four, which we believe has no MDP violations.

**Dialogue Cost Convergence:** We first report on how the dialogue cost convergences as the number of training epochs increases. We learned 10 policies for 4,327,500 epochs of training, with test sessions intermixed, as described in Sec. 3. In order to show how the dialogue cost is converging, we compare the dialogue cost of the learned policies to the optimal cost. Although we cannot be sure what the optimal dialogue cost is, we set it to the lowest cost seen across all 10 policies across all test sessions, which is 6.8260.

Figure 3 gives the results. For each test session, we compute the difference between the dialogue cost and the optimal cost, averaging across the 10 policies and smoothing with the neighboring 15 test sessions before and 15 after. As can be seen, the average dialogue cost is converging to the optimal value. However, even after 4,327,500 of training, it has not converged.

![Figure 3: Dialogue Cost Convergence](image)

**Policy Convergence:** Convergence is not only affecting dialogue cost, but it also affects what the current policy is. How is RL converging on a policy? Will RL ever choose a policy that has worse dialogue performance than one it has already seen? For our car-buying domain, we found that the optimal policy is first seen between 2,000 and 21,500 epochs of training, with an average of 6,450. After RL found an optimal policy, we measured how often RL sees it again in future test sessions. The result is shown in Figure 4 with the line labeled ‘optimal’. We smoothed the curve by averaging the 10 learned policies over neighboring test sessions (15 before and 15 after). As can be seen, once it finds an optimal policy it does not stay there. However, as training progresses, it spends more and more time...
in the optimal policy. We also determined where RL is spending the rest of its time. We measured how often RL is in a policy within 0.1%, 0.3% and 1% of the optimal cost. As shown in Fig. 4, as training progresses, RL spends more of its time closer to the optimal dialogue cost.

**Q Value Convergence:** We now examine how RL converges on the Q value for each state-action pair. We do this by comparing the Q_train and Q_test values, which we introduced in Sec. 4. As in that section, we take the difference between the Q_train and Q_test values for each state-action pair in the current policy during testing. We only examine the state-action pairs in the current policy, as these are the only ones for which we have Q_test scores for. Furthermore, as training progresses, these are the state-action pairs that are in the optimal policy, which are the ones that we care most about.

Figure 5 shows the absolute difference between the Q_train and Q_test values as training increases, across the state-action pairs in the current policy. We average across the 10 learned policies, and smooth with the neighboring test sessions (5 before and 5 after). We show the average difference across all of the states for the 10 learned policies, as well as the average across the 10 learned policies of the maximum difference. As can be seen, as training progresses, the differences decrease.

This study supplements the one on policy convergence as it shows that RL is converging on the actual Q values for each state-action pair. It helps demonstrate that no matter how close two policies are in dialogue cost, RL will eventually find the better one due to convergence at the state-action pair level.

**Exploration and Convergence:** We now investigate how quickly convergence happens in terms of state-action exploration. Although absolute convergence is not achieved, even after 4,327,500 epochs of training, RL has seen the best policy by 6,450 epochs, and is able to stay within 1% after 7,000 epochs. As shown in Fig. 6, RL has seen less than 90% of the total number of state-action pairs. So, RL does not need to have seen every state-action pair before it settles on a good policy.

6. Conclusion

In this paper, we presented two methods for determining whether there is an MDP violation, and if there is, what is its cause. More work is needed to determine their generality.

We also presented a case study in how convergence occurs in our car-buying domain. This study was facilitated by removing two sources of noise: MDP violations, which can prevent convergence, and not using the exact user distribution, which can prevent accurate determination of the actual dialogue cost.

We found that once RL found an optimal policy, it might take a very long time before it consistently returns it. However, the longer training progresses, the more likely it is to return the optimal policy. Furthermore, once RL has seen an optimal policy, it will probably return one fairly close to optimal. Although this does sound negative, one must remember that the cost function used for training a dialogue policy is often not exact [10]. In fact, there might be more noise due to how the cost function was created than from RL taking a long time to converge.

7. Acknowledgment

Funding acknowledged from the National Science Foundation under grants IIS-0713698, IIS-0931338, and IIS-1035156.

8. References