Switch-Based Active Deep Dyna-Q: Efficient Adaptive Planning for Task-Completion Dialogue Policy Learning

Yuexin Wu,* **Xiujun Li,**†‡ **Jingjing Liu,**† **Jianfeng Gao,**† **Yiming Yang***
*Carnegie Mellon University, †Microsoft Research
‡Paul G. Allen School of Computer Science & Engineering, University of Washington
*{yuexinw,yiming}@cs.cmu.edu, †{xiul,jingjl,jfgao}@microsoft.com

Abstract

Training task-completion dialogue agents with reinforcement learning usually requires a large number of real user experiences. The Dyna-Q algorithm extends Q-learning by integrating a world model, and thus can effectively boost training efficiency using simulated experiences generated by the world model. The effectiveness of Dyna-Q, however, depends on the quality of the world model - or implicitly, the pre-specified ratio of real vs. simulated experiences used for Q-learning. To this end, we extend the recently proposed Deep Dyna-Q (DDQ) framework by integrating a *switcher* that automatically determines whether to use a real or simulated experience for Q-learning. Furthermore, we explore the use of active learning for improving sample efficiency, by encouraging the world model to generate simulated experiences in the state-action space where the agent has not (fully) explored. Our results show that by combining switcher and active learning, the new framework named as Switch-based Active Deep Dyna-Q (Switch-DDQ), leads to significant improvement over DDQ and Q-learning baselines in both simulation and human evaluations.[1]

Introduction

Thanks to the increasing popularity of virtual assistants such as Apple's Siri and Microsoft's Cortana, there has been a growing interest in both industry and research community in developing task-completion dialogue systems (Gao, Galley, and Li 2018). Dialogue policies in task-completion dialogue agents, which control how agents respond to user input, are typically trained in a reinforcement learning (RL) setting (Young et al. 2013; Levin, Pieraccini, and Eckert 1997). RL, however, usually requires collecting experiences via direct interaction with real users, which is a costly data acquisition procedure, as real user experiences are much more expensive to obtain in the dialogue setting than that in simulation-based game settings (such as Go or Atari games) (Mnih et al. 2015; Silver et al. 2016).

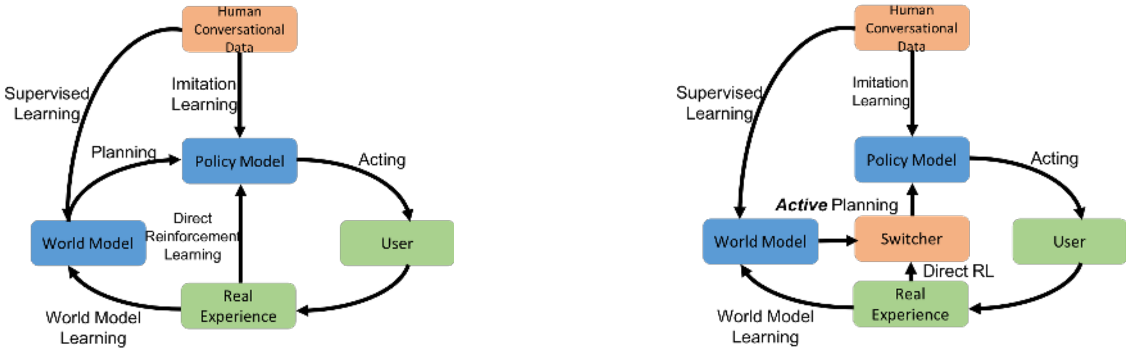
One common strategy is to train policies with user simulators that are developed from pre-collected human-human conversational data (Schatzmann et al. 2007; Li et al. 2016). Dialogue agents interacting with such user simulators do not incur any real-world cost, and can in theory generate unlimited amount of simulated experiences for policy training. The learned policy can then be further fine-tuned using small amount of real user experiences (Dhingra et al. 2016; Su et al. 2016; Lipton et al. 2016; Li et al. 2017).

Although simulated users provide an affordable alternative, they may not be a sufficiently truthful approximation to human users. The discrepancy between simulated and real experiences inevitably leads to strong bias. In addition, it is very challenging to develop a high-quality user simulator, because there is no widely accepted metric to assess the quality of user simulators (Pietquin and Hastie 2013). It remains a controversial research topic whether training agents through user simulators is an effective solution to building dialogue systems.

Recently, Peng et al. (2018) proposed Deep Dyna-Q (DDQ), an extension of the Dyna-Q framework (Sutton 1990), which integrates planning into RL for task-completion dialogue policy learning. As illustrated in Figure 1a, DDQ incorporates a trainable user simulator, referred to as the *world model*, which can mimic real user behaviors and generate simulated experience. The policy of the dialogue agent can be improved through either (1) real user experiences via *direct RL*; or (2) simulated experiences via *indirect RL* or *planning*.

DDQ is proved to be sample-efficient in that a reasonable policy can be obtained using a small number of real experiences, an affordable training process thanks to the integration of planning into RL. However, the effectiveness of DDQ depends, to a large degree, upon the way we control the ratio of real vs. simulated experiences used in different stages of training. For example, Peng et al. (2018) pointed out that although aggressive planning (i.e., policy learning using a large number of simulated experiences) often helps improve the performance in the beginning stage of training when the agent is not sensitive to the low-quality experiences, such aggressive planning might hurt the performance in the later stage when the agent is more susceptible to noise, as illustrated in Figure 2. Carefully designed heuristics are essential to set the ratio properly. For example, we might decrease the number of simulated experiences during the course of training. However, such heuristics can vary with different settings, and thus significantly limits the wide application of DDQ in developing real-world dialogue agents.

[1] Source code is at https://github.com/CrickWu/Switch-DDQ.



(a) DDQ framework

(b) Proposed Switch-DDQ framework

Figure 1: Designs of RL agents for dialogue policy learning in task-completion dialogue systems

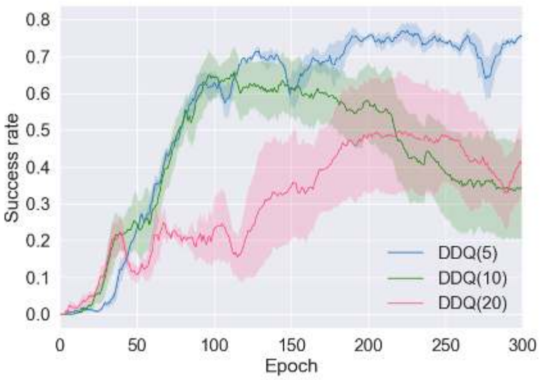


Figure 2: The learning curves of DDQ(K) without heuristics where $(K-1)$ denotes the number of planning steps. The curves are sensitive to K values and may deteriorate in the later phase due to the low-quality simulated experiences.

Another limitation of DDQ is that the world model generates simulated experiences by uniformly sampling user goals. However, training samples in the state-action space unexplored or less explored by the dialogue agent are usually more desirable in order to avoid bias. This is the problem that many active learning methods try to address. In this paper, we present a new variant of DDQ that addresses these two issues.

Our method is inspired by the recent study of Su et al. (2018), which tries to balance the use of simulated and real experience by measuring the quality of simulated experiences using a machine-learned *discriminator*. The more simulated experiences are used if their quality is higher. Their approach demonstrates some limited success, and suffers from two shortcomings. First, it does not take into account the fact that the agent in different training stages might require simulated experiences of different qualities. Second, it still uniformly samples user goals and is not as sample-efficient as it should be (e.g., by using active learning).

In this paper, we propose a new framework, called Switch-based Active Deep Dyna-Q (Switch-DDQ), to significantly improve DDQ's sample efficiency. As illustrated in Figure 1b, we incorporate a *switcher* to automatically determine whether to use real or simulated experiences at different stages of dialogue training, eliminating the dependency on heuristics. The switcher is implemented based on an LSTM model, and is jointly trained with the dialogue policy and the world model. Moreover, instead of randomly sampling simulated experiences, the world model adopts an *active* sampling strategy that generates simulated experiences from the state-action space that has not been (fully) explored by the dialogue agent. Experiments show that this active sampling strategy can achieve a performance that is comparable to the original DDQ method but by using a much smaller amount of real experiences.

The work present in this paper contributes to the growing family of model-based RL methods, and can potentially be applied to other RL problems. To the best of our knowledge, Switch-DDQ is the first learning framework that conducts active learning in a task-completion dialogue setting. The contributions of this work are two-fold:

- We propose a Switch-based Active Deep Dyna-Q framework to incorporate active learning into the Dyna-Q framework for dialogue policy learning, providing a mechanism of automatically balancing the use of simulated and real user experiences.
- We validate the superior performance of Switch-DDQ by building dialogue agents for the movie-ticket booking task. The effectiveness of active learning and switcher is verified by simulation and human evaluations.

Model Architecture

We depict our Switch-DDQ pipeline in Figure 3. The agent consists of six modules: (1) an LSTM-based natural language understanding (NLU) module (Hakkani-Tür et al. 2016) for extracting user intents/goals and calculated their associated slots; (2) a state tracker (Mrkšić et al. 2016) for tracking dialogue states; (3) a dialogue policy that makes choice of the next action by using the information of the current dialogue state; (4) a model-based natural language

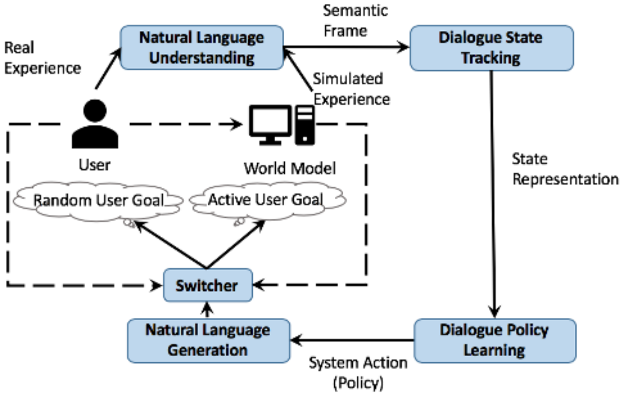


Figure 3: Switch-DDQ for dialogue policy learning.

generation (NLG) module which outputs natural language responses (Wen et al. 2015); (5) a world model for generating simulated user actions and simulated rewards based on active user goal selection; and (6) an RNN-based switcher for selecting the source of data (simulated or real experiences) for dialogue policy training. The solid lines in the figure illustrate the iterative dialogue policy training loop, while the dashed lines show the flow of data in training the world model and switcher.

The optimization of Switch-DDQ comprises four steps: (1) *direct reinforcement learning*: the agent conducts direct interactions with a real user, where the generated real experiences are directly used to improve the dialogue policy; (2) *active planning*: the agent interacts with the simulator and improves the policy using the simulated experiences; (3) *world model learning*: the world model receives real experiences and updates itself; and (4) *switcher training*: the switcher is learned and refined using both real and simulated experiences. Each step is detailed in the subsections below. The iterative Switch-DDQ algorithm, described in pseudocode, is shown in Algorithm 1.

Direct Reinforcement Learning and Planning

Typically, dialogue policy learning can be formulated as a Markov Decision Process in the RL setting, a task-completion dialogue could be viewed as a sequence of (state, action, reward) tuples. We employ the Deep Q-network (DQN) (Mnih et al. 2015) for training the dialogue policy (line 12 in Algorithm 1). Both the direct reinforcement learning and planning are accomplished using the same Q-learning algorithm using the simulated and real experiences, respectively.

Specifically, at each step, the agent receives the state s and selects an action a to carry into the next dialogue turn. The action a is chosen using the exploration policy based on ϵ-greedy, where there is probability ϵ a random action being executed or otherwise the action that maximizes the $Q(s, a; \theta_Q)$ function. The function $Q(\cdot)$ is parameterized by a Multi-Layer Perceptron (MLP) parameterized by θ_Q. Afterwards, the agent observes a reward r from the environment, and a corresponding response a^u from either a real user or the simulator, updating the dialogue state to s' until reaching the end of a dialogue. The experience (s, a, r, a^u, s') is then stored into the user experience buffer B^u or simulator experience buffer B^s respectively. Function $Q(\cdot)$ can be improved using experiences stored in the buffers.

In the implementation, we optimize the parameter θ_Q w.r.t. the mean-squared loss:

$$\mathcal{L}(\theta_Q) = \mathbb{E}_{(s,a,r,s') \sim B^s \cup B^u} \left[(y - Q(s, a; \theta_Q)) \right] \tag{1}$$

$$y = r + \gamma \max_{a'} Q'(s', a'; \theta_{Q'}) \tag{2}$$

where $Q'(\cdot)$ is a copy of the previous version of $Q(\cdot)$ and is only updated periodically and $\gamma \in [0, 1]$ is the discount factor. $Q(\cdot)$ is updated using back-propagation and mini-batch gradient descent.

Algorithm 1 Switch-based Active Deep Dyna-Q

```
1:  procedure SWITCH-DDQ TRAININGPIPELINE
2:      for $i \leftarrow 1$ : max_epoch do
3:          user randomly picks a user goal $g^u$
4:          Generate real experience $e^u$ from user based on $g^u$ into $B^u$
5:          repeat
6:              Actively select a user goal $g^s$ based on the validation results # see Algorithm 2
7:              Generate simulated experience $e^s$ from simulator based on $g^s$ into $B^s$
8:              Evaluate quality of $e^s$ through switcher
9:          until quality $<$ threshold
10:         Train simulator on $B^u$
11:         Train switcher on $B^u, B^s$
12:         Train agent on $B^u, B^s$
13:         Evaluate simulator on validation set
```

Active Planning based on World Model

In a typical task-completion dialogue (Schatzmann et al. 2007), a user begins a conversation with a particular goal in mind G which consists of multiple constraints. For example, in the movie-ticket-booking scenario, the constraints can be the place of the theater, the number of tickets to buy, and the name of the movie. An example of a user goal is `request(theater;numberofpeople=2, moviename=mission_impossible)`, which is presented in its natural language form as "`in which theater can I buy two tickets for mission impossible`". Although there is no explicit restriction for the range of user goals in real experiences, in the stage of planning, the world model can *selectively* generate the simulated experiences in the state-action space that are not (fully) explored by the dialogue agent, based on a specific set of user goals, to improve sample efficiency. We call our planning *active planning* because it is a form of active learning.

The world model for active planning consists of two parts: (1) a user goal sampling module that samples a proper user goal at the start of a dialogue; (2) a response generation module that imitates real users' interaction with the agent to generate for each dialogue turn the user action, reward and the user's decision whether to terminate the dialogue.

- *Active user goal sampling module*. Assume that we have collected large amounts of user goals from human-human conversational data. These user goals can be grouped into different categories, each with different constraints, amounting to different scales of difficulties. The key observation is that, during the training process, while monitoring the performance of the agent policy on validation set, we can gather detailed information about the impact of each category of user goals on the performance improvement of the dialogue agent e.g., in terms of the success rate (line 13 in Algorithm 1). The detailed information can be used to measure the cost (or gain) in the active learning setting (Russo et al. 2018; Auer, Cesa-Bianchi, and Fischer 2002) and guide the world model how to sample user goals.

  Suppose there are k different categories of user goals. At each epoch, the failure rate of each category estimated on the validation set is denoted as f_i and the number of samples for the estimation is n_i. For simplicity, denote the summation of n_i as $N = \sum_i n_i$. Then, the active sampling routine (line 6 in Algorithm 1) can be expanded as

Algorithm 2 Active Sampling Routine

```
procedure ACTIVE USER GOAL SAMPLING
    Draw a number $p_i$ for each category following $p_i \sim \mathcal{N}\left(f_i, \sqrt{\frac{k \ln N}{n_i}}\right)$
    Select the user goal $i$ with the maximum $p_i$ value
```

  Here, $\mathcal{N}$ is the Gaussian distribution for introducing randomness. The Thompson-Sampling-like (Russo et al. 2018) sub-routine of Algorithm 2 is motivated by two observations: (1) on average, categories with larger failure rate f_i are more preferable as they inject more difficult cases (containing more useful information to be learned) based on the current performance of the agent policy. The generated data (simulated experiences) are generally associated with the steepest learning direction and can prospectively boost the training speed; (2) categories that are estimated less reliably (due to a smaller value of n_i value) may have a large de facto failure rate, thus worth being allocated with more training instances to reduce the uncertainty. $\sqrt{\frac{k \ln N}{n_i}}$ is the measurement of the uncertainty of f_i, serving the role of variance in the Gaussian. Thus, the categories with high uncertainty are still likely to be selected even if the failure rate is small.

- *Response generation module*. We utilize the same design of the world model in Peng et al. (2018). Specifically, we parameterize it using a multi-task deep neural network (Liu et al. 2015). Each time the world model observes the dialogue state s and the last action from the agent a, it passes the input pair (s, a) through an MLP $M(s, a; \theta_M)$ generating a user action a^u, a regressed reward r and a binary terminating indicator signal t. The MLP has a common sharing representation in the first layer (referred to as layer h). The computation for each term can be shown as below:

$$h = \tanh(W_h(s, a) + b_h) \tag{3}$$

$$a^u = \texttt{softmax}(W_a h + b_a) \tag{4}$$

$$r = \tanh(W_r h + b_r) \tag{5}$$

$$t = \texttt{sigmoid}(W_t h + b_t) \tag{6}$$

Switcher

At every step of training, the switcher needs to decide whether the dialogue agent should be trained using simulated or real experience (lines 8-9 in Algorithm 1).

The switcher is based on a binary classifier implemented using a Long Short-Term Memory (LSTM) model (Hochreiter and Schmidhuber 1997). Assume that a dialogue is represented as a sequence of dialogue turns, denoted by $\{(s_i, a_i, r_i)\}$, $i = 1, ..., N$, where N is the number of dialogue turns of the dialogue. Q-learning takes a tuple in the form of (s, a, r, s') as a training sample, which can be extracted from two consecutive dialogue turns in a dialogue. Now, the design choice of switcher is whether the classifier is turn-based or dialogue-based. We choose the former, though a bit anti-intuitive, for data efficiency. There is an order of magnitude larger number of turns than that of dialogues. As a result, a turn-based classifier can be more reliably trained than a dialogue-based one. Then, given a dialogue, we score the quality of each of its dialogue turns, and then averages these scores to measure the quality of the dialogue (line 6 in Algorithm 1). If the dialogue-level score is below a certain threshold, the agent switches to interact with real users.

Note that each dialogue turn is scored by taking into account its previous turns in the same dialogue. Given a dialogue turn (s_t, a_t, r_t) and its history $h = ((s_1, a_1, r_1), (s_2, a_2, r_2), ..., (s_{t-1}, a_{t-1}, r_{t-1}))$ We use LSTM to encode h using the hidden state vector, and output a turn-level quality score via a sigmoid layer:

$$\text{Score}((s, a, r), h; \theta) = \texttt{sigmoid}(\text{LSTM}((s, a, r), h; \theta)) \tag{7}$$

Since we store user experiences and simulated experiences in the buffers B^u and B^s, respectively (lines 4, 7 in Algorithm 1), the training of Score(.) follows a similar process of minimizing the cross-entropy loss as in the common domain adversarial training setting (Ganin et al. 2016) using mini-batches:

$$\begin{aligned} &\min_{\theta_S} \mathbb{E}_{(s,a,r),h \sim B^u} \log\left(\text{Score}\left((s, a, r), h; \theta\right)\right) \\ &+ \mathbb{E}_{(s,a,r),h \sim B^s} \log\left(1 - \text{Score}((s, a, r), h; \theta)\right) \end{aligned} \tag{8}$$

Since the experiences stored in B^s and B^u change during the course of dialogue training, the score function of the switcher is updated accordingly, thus automatically adjusting how much planning to perform at different stages of training.

Experiments

We evaluate the proposed Switch-DDQ framework in the movie-ticket booking domain, in two settings: simulation and human evaluation.

Dataset

For experiments, we use a movie-ticket booking dataset which contains raw conversational data collected via Amazon Mechanical Turk. The dataset is manually labeled based on a schema defined by domain experts. As shown in Table 1, the annotation schema consists of 11 intents and 16 slots. In total, the dataset contains 280 labeled dialogues, the average length of which is 11 turns.

	Annotations
Intent	request, inform, deny, confirm_question, confirm_answer, greeting, closing, not_sure, multiple_choice, thanks, welcome
Slot	city, closing, date, distanceconstraints, greeting, moviename, numberofpeople, price, starttime, state, taskcomplete, theater, theater_chain, ticket, video_format, zip

Table 1: The data annotation schema

Baselines

We compare the effectiveness of the Switch-DDQ agent with several baselines:

- **DQN** agent is implemented with only direct reinforcement learning in each training epoch (without lines 5-9 in Algorithm 1).
- The **DQN(**K**)** has $(K-1)$ times more real experiences than the DQN agent (repeat lines 3-4 in Algorithm 1 K times). The performance of DQN(K) can be viewed as the upper bound of DDQ (K), with the same number of planning steps $(K-1)$ (they have the same training setting and the same amount of training samples during the entire learning process).
- The **DDQ(**K**)** agents are learned using a jointly-trained world model initiated from human conversational data, with $(K-1)$ planning steps (replace lines 5-9 in Algorithm 1 with a $(K-1)$-round loop).
- The proposed **Switch-DDQ** agents are updated as described in Algorithm 1. Note that there is no parameter K in the agent, as real/simulated ratio is automatically controlled by the switcher module.

Implementation Details

Agent and Hyper-parameter Settings We use an MLP to parameterize function $Q(\cdot)$ in all the agent variants (DQN, DDQ and Switch-DDQ). The MLP has one hidden layer of 80 neurons with ReLU (Nair and Hinton 2010) activation function. The ϵ-greedy policy is adopted to explore the action space. The discount factor γ for future rewards is set to 0.9. For DDQ(K), as the number of real and simulated experiences is different at each epoch, the buffer sizes of B^u and B^s are generally set to 2000 and $2000 \times K$, respectively. For Switch-DDQ, we observed that the results are not sensitive to the buffer size of B^s, so we set it to 2000×5 for all settings.

We randomly initialize the parameters in all neural networks and empty both experience buffers B^u and B^s in the beginning. The RMSProp (Hinton, Srivastava, and Swersky 2012) algorithm is used to perform optimization over all the parameters where the learning rate is set to 0.001. We also apply the gradient clipping trick to all parameters with a maximum norm of 1 to prevent possible gradient explosion issues. At the beginning of each epoch (line 2 in Algorithm 1), the reference copy $Q'(\cdot)$ is updated. Each simulated dialogue contains less than 40 turns. Conversations exceeding the maximum number of turns are counted as failed. In order to train the agents more efficiently, we utilized the imitation learning method called Reply Buffer Spiking (RBS) (Lipton et al. 2016) at the initial stage to build a simple rule-based agent trained from human conversational data. The trained agent is then used to pre-fill the real experience replay buffer B^u with a total of 50 complete dialogues before training all the variants of the agent.

World Model We employ an MLP world model for DDQ and Switch-DDQ. The shared hidden layer is set to have size 160 with hyperbolic tangent activation. The state and action input are encoded through a linear layer of size 80. We pre-fill each n_i as 5 to prevent division by 0 error, during the calculation of the Gaussian variance (line 2 in Algorithm 2).

Switcher The LSTM switcher has a hidden layer with 126 cells. Similar to the world model, states and actions are passed through a linear layer of size 80 as inputs at each time step. The switcher adopts an annealing threshold w.r.t. the epoch number to decide the quality of each dialogue turn. If the average dialogue episode score passes a certain threshold, all the high-quality predictions are pushed into buffer B^s.[2]

Simulation Evaluation

We train the dialogue agents by simulating interactions between the agents and well-programmed user simulators, instead of real users. That is, we train the world model to imitate the behaviors of the user simulator.

User Simulator We used an open-sourced task-oriented user simulator (Li et al. 2016) in our simulated evaluation. At each dialogue turn, the simulator will emit a simulated user response to the agent. When the dialogue ends, a reward signal will be provided. The dialogue is considered successful, if and only if a movie ticket is booked successfully and the information provided by the agent conform to all the constraint slots in the sampled user goal. Each completed dialogue shows either a positive reward $2 * L$ for success, or a negative reward $-L$ for failure, where L is the maximum

[2] See the code for specific hyper-parameters.

Agent	Epoch = 100			Epoch = 200			Epoch = 300		
	Success	Reward	Turns	Success	Reward	Turns	Success	Reward	Turns
DQN	0.2867	-17.35	25.51	0.6733	32.48	18.64	0.7667	46.87	12.27
DQN(5)	*0.7667*	*46.74*	*12.52*	*0.7867*	*49.46*	*11.88*	*0.8000*	*50.81*	*12.37*
DDQ(5)	0.6200	25.42	19.96	0.7733	45.45	16.69	0.7467	43.22	14.76
DDQ(10)	**0.6800**	34.42	16.36	0.6000	24.20	17.60	0.3733	-2.11	15.81
DDQ(20)	0.3333	-13.88	29.76	0.4467	5.39	18.41	0.3800	-1.75	16.69
Switch-DDQ	0.5200	15.48	15.84	**0.8533**	56.63	13.53	**0.7800**	48.49	12.21

Table 2: Results of different agents at training epoch $= \{100, 200, 300\}$. Each number is averaged over 3 runs, and each run is tested on 50 dialogues. (Success: success rate) Switch-DDQ outperforms DQN and DDQ variants after Epoch 100, where DQN(5) is shown as the upper bound as it uses more real experiences. Best scores are labeled in bold faces.

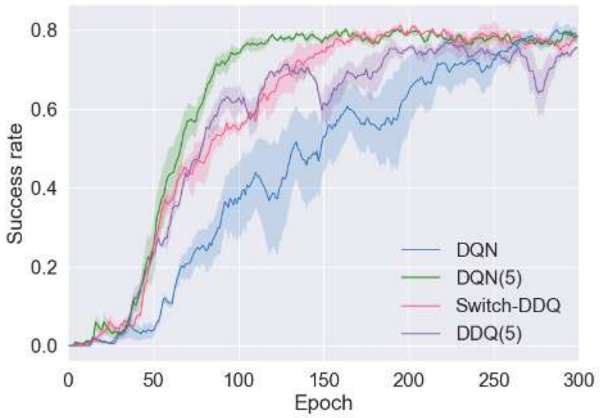


Figure 4: The learning curves of DQN, DQN(5), Switch-DDQ, and DDQ(5) of each epoch.

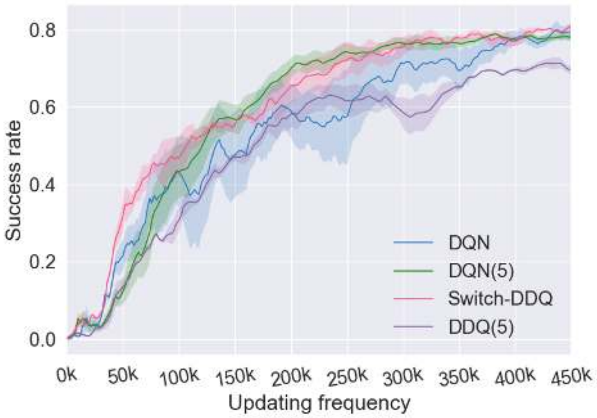


Figure 5: The learning curves of DQN, DQN(5), Switch-DDQ, and DDQ(5) on the scale of updating frequency.

number of turns in each dialogue and is set to 40 in our experiments. Furthermore, in each turn, a negative reward -1 is provided to encourage shorter dialogue.

Main Results We summarize the main results in Table 2 and plot the learning curves in Figure 4. As illustrated in Figure 2, DDQ(K) is highly susceptible to parameter K. Therefore, we only keep the best performing DDQ(5) as the baseline in the following figures. DQN(5), which uses 4 times more real user experiences to this end, is the upper bound for the corresponding DDQ(5) method. In Table 2, we report success rate, average reward and average number of turns over 3 different runs for each agent. As is shown, the agent of Switch-DDQ after the first 100 epochs, consistently achieves higher success rates with a smaller number of interaction turns. Again, DDQ(10) and DDQ(20) quickly deteriorate through the training process. In Figure 4, we can observe that in the first 130 epochs, DDQ(5) performs slightly better than Switch-DDQ. However, after that, Switch-DDQ surpasses DDQ(5) and achieves better performance. It only takes Switch-DDQ 180 epochs to achieve comparable results to DQN(5), which utilizes 4 times more real experiences, and DDQ(5) fails to reach similar performance within 300 epochs. This is expected, as the aggressive simulator sampling policy adopted by DDQ(5), though helping update the policy network more rapidly in the early stage of training, hurts the performance due to the use of low-quality training instances in the later stage. Note that except for DQN(5), all the agents are trained using the same number of real experiences in each epoch, differing only the amounts of simulated experiences used (for planning) and how these simulated experiences are generated (via active learning or not). The result show that Switch-DDQ can utilize simulators in a more effective and robust way than DDQ.

We also examine the performance of different agents with an equal number of optimization operations. As shown in Figure 5, we plot the success rate as a function of updating frequency, i.e., how many dialogue experiences (either real or simulated) are used altogether to optimize the agent policy network. Note that DQN(5) displays superior performance over DQN as it generates more diverse dialogues at the same updating frequency (DQN may refer to identical experiences more frequently since B^u in DQN is refreshed less often than that in DQN(5)). Furthermore, we observe that DDQ(5) fails to obtain a similar performance to DQN, due to the use of many low-quality simulated experiences. However, this does not happen in Switch-DDQ, since it actively samples user goals by making diversified training di-

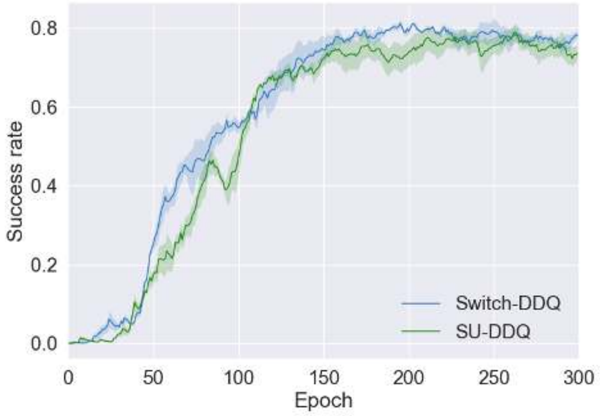


Figure 6: Learning curves of Switch-DDQ versus SU-DDQ where SU-DDQ uses a uniform sampling strategy.

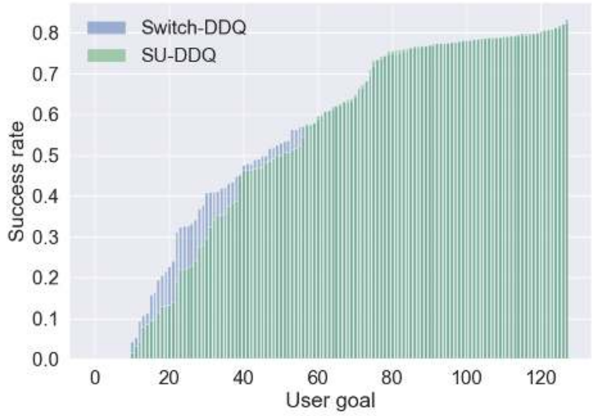


Figure 7: Success rate on 128 user goal categories for Switch-DDQ and SU-DDQ, ranking in ascending order.

alogues and discreetly controlling the amount of simulated experiences via the switcher.

Ablation Test To further examine the effectiveness of the active learning module, we conduct an ablation test by replacing the user goal selection routine (Algorithm 2) with the one based on uniform sampling, referred to as SU-DDQ. The results in Figure 6 demonstrate that Switch-DDQ can consistently outperform SU-DDQ, especially in the early phase (before epoch 100). This is due to the fact that the agent is more sensitive to the diversity of user goals in the earlier stage since in the limited data setting, many repeated cases introduce biases more easily. In Figure 7, we report the success rate for different categories of user goals and rank them in the increasing order. It is observed that for the corresponding rank of user goal category, especially the ones with low success rate, the active version of Switch-DDQ always give a better score. These results demonstrate that the use of the active module improves training efficiency.

Human Evaluation

Real users were recruited to interact with different agents, while the identity of the agent system is hidden from the users. At the beginning of the dialogue session, the user was provided with a randomly sampled user goal, and one of the agents was randomly picked to converse with the user. The dialogue session can be terminated at any time, if the user finds that the dialogue takes so many turns that it is unlikely to reach a promising outcome. Such dialogues are considered as failed in our experiments.

Three agents (DQN, DDQ(5), and Switch-DDQ) trained as previously described (Figure 4) at epoch 150 are selected as for human evaluation.[3] As illustrated in Figure 8, the results of human evaluation are consistent with those in the simulation evaluation. We find that DQN is abandoned more often by users as it takes so many dialogue turns (Table 2) resulting in a much hefty performance drop, and the proposed Switch-DDQ outperforms all the other agents.

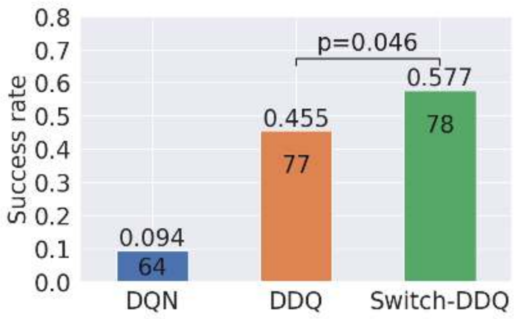


Figure 8: Human evaluation results of DQN, DDQ(5), and Switch-DDQ. The number of test dialogues is shown on each bar, and the one-sided p-value is from a two-sample permutation test over the success/fail lists.

Conclusion

This paper presents a new framework Switch-based Active Deep Dyna-Q (Switch-DDQ) for task-completion dialogue policy learning. With the introduction of a switcher, Switch-DDQ is capable of adaptively choosing the proper data source to use, either from real users or world model, enhancing the efficiency and robustness of dialogue policy learning. Furthermore, the active user goal sampling strategy provides a better utilization of the world model than that of previous DDQ, and boosts the performance of training. Validating Switch-DDQ in the movie-ticket booking task with simulation experiments and human evaluation, we show that the Switch-DDQ agent outperforms the agents trained by other state-of-the-art methods, including DQN and DDQ. Switch-DDQ can be viewed as a generic model-based RL approach, and is easily extensible to other RL problems.

[3] Epoch 150 is picked since we are testing the effectiveness of methods using a small number of real experiences.

Acknowledgement

We thank the reviewers for their helpful comments, and we would like to acknowledge the volunteers for helping us with the human experiments. This work was done in part when Yuexin Wu was visiting Microsoft Research as an intern, and is supported in part by the National Science Foundation (NSF) under grant IIS-1546329.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Dhingra, B.; Li, L.; Li, X.; Gao, J.; Chen, Y.-N.; Ahmed, F.; and Deng, L. 2016. Towards end-to-end reinforcement learning of dialogue agents for information access. *arXiv preprint arXiv:1609.00777*.

Ganin, Y.; Ustinova, E.; Ajakan, H.; Germain, P.; Larochelle, H.; Laviolette, F.; Marchand, M.; and Lempitsky, V. 2016. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research* 17(1):2096–2030.

Gao, J.; Galley, M.; and Li, L. 2018. Neural approaches to conversational ai. *arXiv preprint arXiv:1809.08267*.

Hakkani-Tür, D.; Tür, G.; Celikyilmaz, A.; Chen, Y.-N.; Gao, J.; Deng, L.; and Wang, Y.-Y. 2016. Multi-domain joint semantic frame parsing using bi-directional rnn-lstm. In *Interspeech*, 715–719.

Hinton, G.; Srivastava, N.; and Swersky, K. 2012. Rmsprop: Divide the gradient by a running average of its recent magnitude. *Neural networks for machine learning, Coursera lecture 6e*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Levin, E.; Pieraccini, R.; and Eckert, W. 1997. Learning dialogue strategies within the markov decision process framework. In *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, 72–79. IEEE.

Li, X.; Lipton, Z. C.; Dhingra, B.; Li, L.; Gao, J.; and Chen, Y.-N. 2016. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*.

Li, X.; Chen, Y.-N.; Li, L.; Gao, J.; and Celikyilmaz, A. 2017. End-to-end task-completion neural dialogue systems. *arXiv preprint arXiv:1703.01008*.

Lipton, Z. C.; Gao, J.; Li, L.; Li, X.; Ahmed, F.; and Deng, L. 2016. Efficient exploration for dialogue policy learning with bbq networks & replay buffer spiking. *arXiv preprint arXiv:1608.05081*.

Liu, X.; Gao, J.; He, X.; Deng, L.; Duh, K.; and Wang, Y.-Y. 2015. Representation learning using multi-task deep neural networks for semantic classification and information retrieval.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Mrkšić, N.; Séaghdha, D. O.; Wen, T.-H.; Thomson, B.; and Young, S. 2016. Neural belief tracker: Data-driven dialogue state tracking. *arXiv preprint arXiv:1606.03777*.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.

Peng, B.; Li, X.; Gao, J.; Liu, J.; Wong, K.-F.; and Su, S.-Y. 2018. Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning. In *ACL*.

Pietquin, O., and Hastie, H. 2013. A survey on metrics for the evaluation of user simulations. *The knowledge engineering review* 28(1):59–73.

Russo, D. J.; Van Roy, B.; Kazerouni, A.; Osband, I.; Wen, Z.; et al. 2018. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning* 11(1):1–96.

Schatzmann, J.; Thomson, B.; Weilhammer, K.; Ye, H.; and Young, S. 2007. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, 149–152. Association for Computational Linguistics.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T. P.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Su, P.-H.; Gasic, M.; Mrksic, N.; Rojas-Barahona, L.; Ultes, S.; Vandyke, D.; Wen, T.-H.; and Young, S. 2016. Continuously learning neural dialogue management. *arXiv preprint arXiv:1606.02689*.

Su, S.-Y.; Li, X.; Gao, J.; Liu, J.; and Chen, Y.-N. 2018. Discriminative deep dyna-q: Robust planning for dialogue policy learning. *arXiv preprint arXiv:1808.09442*.

Sutton, R. S. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*. Elsevier. 216–224.

Wen, T.-H.; Gasic, M.; Mrksic, N.; Su, P.-H.; Vandyke, D.; and Young, S. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*.

Young, S. J.; Gasic, M.; Thomson, B.; and Williams, J. D. 2013. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE* 101(5):1160–1179.