Task-Completion Dialogue Policy Learning via Monte Carlo Tree Search with Dueling Network

Sihan Wang, Kaijie Zhou*, Kunfeng Lai, Jianping Shen
Ping An Life Insurance of China, Ltd.
{wangsihan088, zhoukaijie002}@pingan.com.cn
{laikunfeng597, shenjianping324}@pingan.com.cn

Abstract

We introduce a framework of Monte Carlo Tree Search with Double-q Dueling network (MCTS-DDU) for task-completion dialogue policy learning. Different from the previous deep model-based reinforcement learning methods, which uses background planning and may suffer from low-quality simulated experiences, MCTS-DDU performs decision-time planning based on dialogue state search trees built by Monte Carlo simulations and is robust to the simulation errors. Such idea arises naturally in human behaviors, e.g. predicting others' responses and then deciding our own actions. In the simulated movie-ticket booking task, our method outperforms the background planning approaches significantly. We demonstrate the effectiveness of MCTS and the dueling network in detailed ablation studies, and also compare the performance upper bounds of these two planning methods.

1 Introduction

Designing a task-completion dialogue system has become an important task due to its huge commercial values. The dialogue agent aims to help users to complete a single or multi-domain task, e.g. booking a flight and making a hotel reservation. The core of such a system is the dialogue policy module, which enables the agent to respond properly and provide users with the desired information. Early work has shown that dialogue policy learning can be designed as a Markov Decision Process (MDP) (Singh et al., 2002; He et al., 2018; Zhao et al., 2019; Takanobu et al., 2019). Reinforcement learning (RL) is a common framework to solve MDP but it requires huge amounts of interactions with real users, which is generally infeasible in the real world. One way to work around this problem is by designing a user simulator using the real human conversation data (Schatzmann et al., 2007; Li et al., 2016).

Recently, following these ideas, many researchers have applied deep model-based RL methods to task-completion dialogue policy learning (Peng et al., 2018; Su et al., 2018; Wu et al., 2018). In a model-based method, the agent not only updates its action value or policy function through real experiences but also learns how the environment produces the next states and rewards. The learned environment is called a *model*, which can be further used to generate simulated experiences. Using both real and simulated experiences is referred to as *background planning* and can substantially improve the learning efficiency (Sutton and Barto, 2018). Peng et al. (2018) extend Dyna-Q (Sutton, 1990; Sutton et al., 2012) to Deep Dyna-Q (DDQ) for dialogue policy learning and achieve appealing results. However, since model learning cannot be perfect, some simulated experiences with large errors may hinder policy learning (Su et al., 2018). Su et al. (2018) propose to train a discriminator to filter low-quality simulated experiences. Wu et al. (2018) design a switcher-based mechanism to automatically balance the use of real and simulated experiences. Nevertheless, the overall improvement is still limited.

In this paper, we first upgrade the common baseline model of the task-completion dialogue policy learning problem, Deep Q-network (DQN) (Mnih et al., 2015) by adopting its two variants: Deep Double Q-networks (DDQN) (Van Hasselt et al., 2016) and Dueling network (Wang et al., 2015). The purpose is to fully exploit the advanced value-based methods that are orthogonal to planning. We show that the new baseline can achieve comparable performance with DDQ. To further boost the performance, we propose to use Monte Carlo Tree Search (MCTS) (Chaslot et al., 2008) as *decision-*

*Corresponding author

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 3461–3471,
November 16–20, 2020.

time planning (Sutton and Barto, 2018). The differences between background and decision-time planning are illustrated in Figure 1 and 2. Decision-time planning doesn't use the model to generate simulated experiences. In the testing stage or a real decision time, rather than directly picking actions based on action values, a rollout algorithm like MCTS uses the model to build a search tree by running simulations. Performing policy evaluation on the search tree generally yields more accurate estimations of action values assuming that the model is correct.

Due to this property, MCTS has achieved huge success in the game of Go (Silver et al., 2016, 2017). However, its applications in the non-gaming settings, such as dialogue systems, are still rare and little studied. One difficulty is that the model now has to learn the more complex dynamics of state transitions than the deterministic game rules. Consequently, MCTS may grow a erroneous search tree and the resulting estimated action values may be wrong.

To alleviate simulation errors, we design a new MCTS method incorporating the DDQN object and the dueling architecture. The main idea is to focus on the more promising parts of the state-action space and reduce the rollout depths. The dueling network can be used as heuristic or scoring functions in this case. Given dialogue states, it outputs two streams of data: action advantages and state values. Action advantages can be viewed as the prior knowledge to differentiate actions. State values can be used as the approximated rollout results. We denote an agent under this design as MCTS-DDU. Experiments show that MCTS-DDU agent outperforms previous methods by a wide margin in both task success rate and learning efficiency.

Briefly, the main contributions of our work are in the following aspects:

- For the direct reinforcement learning of dialogue policy, we show that an agent trained by the extensions of DQN can perform comparatively with the latest deep model-based methods, which can serve as an advanced baseline for future works.
- For the planning part, we propose to incorporate MCTS with DDQN and Dueling network which exceeds previous approaches significantly. To our best knowledge, we are the first to apply decision-time planning and adapt MCTS for this task.

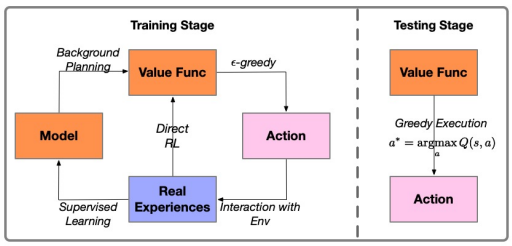

Figure 1: Training and testing stages of value-based background planning.

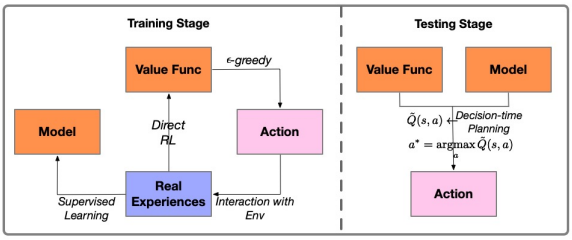

Figure 2: Training and testing stages of decision-time planning.

2 Background

2.1 Reinforcement Learning for Task-Completion Dialogue

Reinforcement learning is a framework to solve sequential decision problems. The problem can be formalized as a Markov Decision Process $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ is a finite action space, $\mathcal{P}$ is a state transition function, $\mathcal{R}$ is a reward function, and γ is a discount factor. A value-based agent aims to learn an action value function as its implicit policy, so that its expected long-term rewards are maximized. Next, we show how to formalize a task-completion dialogue session as a MDP.

State s_t is defined as the dialogue history of previous t turns, containing user intents, associated slots, and agent responses.

Action a_t is defined as dialog act, $(intent, slot)$, representing the agent's intent on a specific slot. Take movie-ticket booking as an example, *(request, #tickets)* means that the agent asks the user how many tickets are needed.

Transition $\mathcal{P}$ represents the dialogue state updates according to stochastic responses from the user to the agent. In the case of a user simulator, the handcrafted rules define the state transitions implicitly.

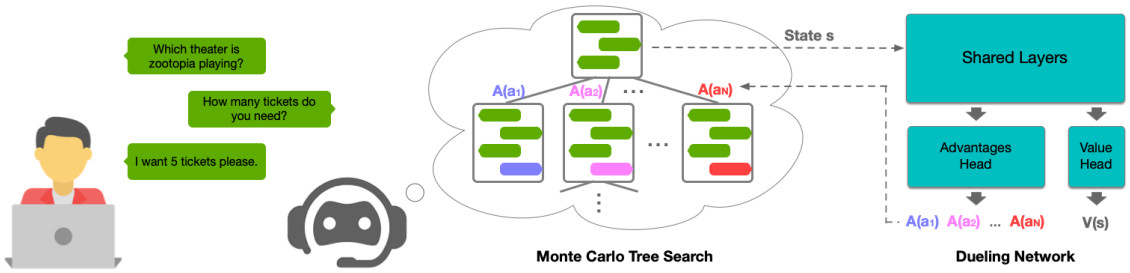

Figure 3: MCTS with dueling network as decision-time planning.

Reward $\mathcal{R}$ is the immediate feedback signal after the agent takes an action to the user, which generally depends on the dialogue status, such as in-process, success, or failure.

2.2 Deep Q-networks and Variants

Deep Q-networks (DQN) DQN combines q-learning (Watkins and Dayan, 1992) with a deep network, noted as $Q(s, a; \theta)$, to approximate state-action values. Generally speaking, training a deep neural network as a value function approximator trends to be notoriously unstable and has no convergence guarantee. To mitigate this problem, Mnih et al. (2015) utilize the *experience replay* technique (Lin, 1991) to reduce data correlation and improve data efficiency. Another critical trick is to maintain a separate *target network* $Q(s, a; \theta^-)$, whose outputs serve as target values, and the parameters θ^- get soft-updated towards θ periodically. To update $Q(s, a; \theta)$, the loss function is defined as:

$$L(\theta) = \mathbb{E}_{e\sim\mathcal{D}}[(y - Q(s_t, a_t; \theta))^2] \quad (1)$$
$$y = r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta^-)$$

where $\mathcal{D}$ is the replay buffer holding experiences $e = (s_t, a_t, r_{t+1}, s_{t+1})$ and γ is the discount factor.

Double Deep Q-networks (DDQN) Q-learning and the plain DQN have the maximization bias problem (Sutton and Barto, 2018). Since the action selection and evaluation are coupled via the maximization operator, $Q(s, a; \theta)$ trends to produce overoptimistic estimations (Hasselt, 2010). Extending the idea of double q-learning (Hasselt, 2010) to the deep reinforcement learning settings, DDQN proposes an alternate loss to use:

$$L(\theta) = \mathbb{E}_{e\sim\mathcal{D}}[(y - Q(s_t, a_t; \theta))^2] \quad (2)$$
$$\hat{a}_{t+1} = \operatorname*{argmax}_a Q(s_{t+1}, a; \theta)$$
$$y = r_{t+1} + \gamma Q(s_{t+1}, \hat{a}_{t+1}; \theta^-)$$

Dueling networks Dueling network is proposed as a novel architecture design for DQN. The network architecture is shown in Figure 3. Given a state, the shared layers generate a compact hidden representation. Then, instead of estimating action values directly, the computation is separated into two streams: state value and action advantages, according to $Q(s, a) = V(s) + A(s, a)$ (Baird III, 1993). This decomposition brings several benefits, such as the improved training efficiency and the separate access to state values and action advantages.

2.3 Planning

Besides directly improving the action value function, real experiences can also be used to learn a model $\mathcal{M} = (\mathcal{P}, \mathcal{R})$, where $\mathcal{P}$ and $\mathcal{R}$ are defined in Section 2.1. Planning refers to utilizing $\mathcal{M}$ to further improve the policy (Sutton and Barto, 2018). Background planning uses $\mathcal{M}$ to generate simulated experiences. By doing this, more data is available for learning. Dyna-Q, DDQ (Peng et al., 2018), D3Q (Su et al., 2018), and Switch-DDQ (Wu et al., 2018) all fall into this category. In contrast, decision-time planning focuses on how to pick an action for a specific state. Namely, it tries to solve a sub-MDP starting from the "current" state. Monte Carlo Tree Search (MCTS) is a kind of decision-time planning algorithm. It uses the model to grow search trees and continually simulate more promising trajectories. Appropriate methods, like Monte Carlo and temporal-difference learning, can be then applied on the search tree to select the best action.

3 Methodology

For a dialogue process involving complex state dynamics, applying planning directly is problematic, since the model learning cannot be perfect. This

may result in low-quality simulated experiences for Dyna-Q and an erroneous search tree for MCTS. In the former case, it is inevitable that parameter updates would be made in some wrong directions for the action value network. While for MCTS, it is possible to reduce the incorrect portions by trimming the depths based on value function that summarize the subtrees, and preferring the actions with higher advantages for branch exploration. Since a value-based agent picks actions based on the action advantages, the state transitions induced by the higher advantages are more likely to be generalized well. Thus, branch exploration with the higher action advantages contains fewer errors by focusing more on the neighborhood of those well learned state space.

Our framework for task-completion dialogue policy learning is presented in Figure 3 and Algorithm 1. In the training stage, the action value network or Q-network $Q = (A, V)$ is optimized via direct reinforcement learning and the model $\mathcal{M} = (\mathcal{P}, \mathcal{R})$ is optimized via model learning respectively. In the testing stage, the agent take actions in a more thoughtful way by performing MCTS with the action advantage head A and the state value head V.

Direct reinforcement learning In this stage, the agent interacts with an user, receives real experiences of the next states and rewards, and optimizes $Q(s, a; \theta, \phi)$ based on the DDQN objective (Eq.2). The reward function works as follows: in each step, the agent receives a penalty of -1. By the end of a dialogue session with the maximal dialogue turns L, the agent receives a reward of $2*L$ if the task is completed successfully or a reward of $-L$ if the task fails. Note that, no planning is executed during this stage, actions are chosen using the ϵ-greedy strategy. Concretely, $a^* = \text{argmax}_a Q(s, a)$ with probability $1 - \epsilon$ and $a^* = uniform(\mathcal{A})$ with probability ϵ.

Model learning The model $\mathcal{M}(s, a; \alpha, \beta) = (\mathcal{P}(s, a; \alpha), \mathcal{R}(s, a; \beta))$ is trained via supervised learning based on pairs $\{(s_t, a_t, s_{t+1})\}$, $\{(s_t, a_t, r_{t+1})\}$ sampled from the replay buffer. We design $\mathcal{M}$ to be a *sample model*, whose $\mathcal{P}(s, a; \alpha)$ produces a sample of s_{t+1} not a distribution over all the next states. By such design, the modelings of user behaviors and state updating are combined in an end-to-end manner. For the transition loss, we use the l_2-norm of the representational differences between s_t and s_{t+1}. For the reward loss, we use the regular regression loss, Mean-Square-Error(MSE).

$$L_{\mathcal{P}}(\alpha) = \mathbb{E}_{e\sim\mathcal{D}}[\|\mathcal{P}(s_t, a_t; \alpha) - s_{t+1}\|_2^2] \quad (3)$$

$$L_{\mathcal{R}}(\beta) = \mathbb{E}_{e\sim\mathcal{D}}[(\mathcal{R}(s_t, a_t; \beta) - r_{t+1})^2] \quad (4)$$

Monte Carlo Tree Search with Dueling network In MCTS, each node represents a state and each edge represents an action causing the state transition. Each edge also stores the statistics of a cumulative action value $Q_c(s, a)$ and a visit count $N(s, a)$. There are four steps in one MCTS simulation process, including *selection*, *expansion*, *simulation* and *backpropagation* (Chaslot et al., 2008). To be more specific, we use the *Upper Confidence Bounds for Tree(UCT)* (Kocsis and Szepesvári, 2006; Kocsis et al., 2006) among the MCTS family.

As mentioned, using an approximated complex environment to simulate with large branch factors and depths may lead to both high bias and high variances problems. To address these issues, the dueling network can assist the plain MCTS by providing 1) normalized action advantages as breadth-wise priorities for exploration and 2) state value estimation as depth-wise early stop, both of which essentially prune the enormous state-action space. Formally, we incorporate UCT with dueling architecture and propose a new upper confidence bound called UCT_D:

$$\text{UCT}_D(s, a) = \frac{Q_c(s, a)}{N(s, a)} + c \cdot A(s, a) \cdot \sqrt{\frac{2 \ln N(s)}{N(s, a)}}$$

where $N(s) = \sum_a N(s, a)$ is the sum of visit counts of all available actions. The first term helps to track the action with the highest empirical action value. The second term encourages the search process to explore the actions with higher normalized advantages or lower visit counts. The constant c is the hyperparameter balancing exploitation and exploration. Silver et al. (2016) use a policy network to produce a prior probability and formulate the second term as $\frac{P(s,a)}{1+N(s,a)}$. The key difference from our method is that: policy network is trained by a policy gradient method (Sutton et al., 2000), which trends to concentrate on the right action given a state but neglects the differences among the rest actions. Next, we will describe the simulation process in detail.

Selection Given a state as the root, the action with the highest UCT_D score is chosen on each tree level. This selection process runs recursively until a not fully expanded node, whose children nodes haven't been expanded all.

Expansion Once such node is reached and the action is again picked by UCT_D, the model $\mathcal{M}$ would produce a leaf node s_L that represents the next state. The reward is stored for that edge and would be used in the backpropagation step.

Simulation In this step, unlike the conventional rollout strategies , we simply use the value head $V(s;\theta)$ of $Q(s,a;\theta,\phi)$ to estimate the state value of s_L as $v(s_L)$. This approach has proved to be effective due to using a deep network as the value function and also efficient since no single rollout is played (Silver et al., 2017).

Backpropagation When the simulation step is finished, $v(s_L)$ is backpropagated upwards through all the ancestor edges and updates the corresponding statistics $Q_c(s,a)$ and $N(s,a)$. The update rules are as follows:

$$N(s,a) \leftarrow N(s,a) + 1 \tag{5}$$

$$\Delta Q_c(s,a) \leftarrow r(s,a) + \gamma \Delta Q_c(s',a') \tag{6}$$

where (s',a') is the child edge of (s,a). The update value for the last edge (s_{L-1},a_{L-1}) is defined as: $\Delta Q_c(s_{L-1},a_{L-1}) \leftarrow r(s_{L-1},a_{L-1}) + \gamma V(s_L)$.

4 Experiments

In this section, we first introduce the experimental setup and baseline models. We also propose a new model-free baseline based on the recent extensions of DQN. The effectiveness of MCTS, advantage function, and DDQN objective are demonstrated via thorough ablation studies. We also explore the tradeoff between exploitation and exploration in MCTS. Lastly, we compare the performance upper bounds of background and decision-time planning with a perfect model.

4.1 Setup and Baselines

We consider the movie-ticket booking task that has been studied in Peng et al. (2018), Su et al. (2018) and Wu et al. (2018). Li et al. (2016) convert 280 real dialogues from Amazon Mechanical Turk to a user goal set $\mathcal{G}$ and a dialogue schema containing 11 *intents* and 16 *slots*, which defines the feasible actions for both users and the agent. Evaluation

Algorithm 1 MCTS with Double-q and Dueling Network for Task-Completion Dialogue Policy Learning

Initialize q-network $Q = (V(s;\theta), A(s,a;\phi))$
Initialize target network: $\theta^- = \theta, \phi^- = \phi$
Initialize model $\mathcal{M} = (\mathcal{P}(s,a;\alpha), \mathcal{R}(s,a;\beta))$
Initialize user goals set $\mathcal{G}$
while True **do**
  Sample a user goal from $\mathcal{G}$ ▷ Training
  Initialize s_1 from user first utterance
  for $t = 1, T$ **do**
    $a_t \leftarrow \epsilon\text{-}greedy(Q(s_t,\cdot;\theta,\phi))$
    Execute a_t and observe s_{t+1}, r_{t+1}
    Store experience $(s_t, a_t, s_{t+1}, r_{t+1})$
  end for
  Optimize $Q(s,a;\theta,\phi)$ based on Eq.(2)
  Optimize $\mathcal{M}(s,a;\alpha,\beta)$ based on Eq.(3-4)
  Update $\theta^- = \tau * \theta^- + (1-\tau) * \theta$,
  $\phi^- = \tau * \phi^- + (1-\tau) * \phi$

  Sample a user goal from $\mathcal{G}$ ▷ Testing
  Initialize s_1 as the root
  for t $= 1, T$ **do**
    for Simulation $= 1, M$ **do** ▷ MCTS
      $s \leftarrow s_t$
      while s is fully expanded **do**
        $a' \leftarrow \text{argmax}_a \text{UCT}_D(s,a)$
        $s \leftarrow \mathcal{P}(s,a';\alpha)$
      end while
      $a' \leftarrow \text{argmax}_a \text{UCT}_D(s,a)$
      Expand leaf state $s_L \leftarrow \mathcal{P}(s,a';\alpha)$
      Estimate $v(s_L) \leftarrow V(s_L;\theta)$
      while the root s_t is not reached **do**
        Update statistics $Q_c(s,a)$ and
        $N(s,a)$ based on Eq.(5-6)
      end while
    end for
    $a_t \leftarrow \text{argmax}_a \frac{Q_c(s_t,a)}{N(s_t,a)}$
    Execute a_t and observe s_{t+1}, r_{t+1}
  end for
end while

Figure 4: The learning curves of DDQ(K).

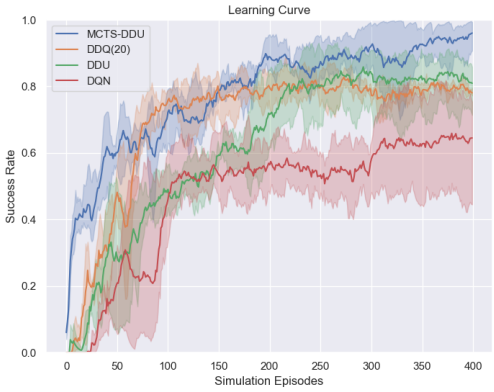

Figure 5: The learning curves of MCTS-DDU, DDU, DDQ(20) and DQN.

metric is the task success rate. The task is considered as completed successfully only when the agent manages to provide all of the users' desired information, propose appropriate suggestions, and finally inform the booking.

In the training stage, we use the user simulator implemented by Li et al. (2016) as the environment. For a dialogue session, the simulator first samples a goal from the goal set $\mathcal{G}$ and generate the first utterance. Once the conversation begins, the user simulator would make responses based on the predefined rules and agent's replies. Rewards are provided to the agent based on the dialogue status (as described in the part of Direct reinforcement learning).

We also use the strategy called *Replay Buffer Spiking* proposed in Lipton et al. (2018), to prefill the experience replay buffer by allowing a rule-based agent to interact with the user simulator. The successful experiences executed by the rule-based agent could considerably speed up the following training stage. Otherwise, it may take thousands of episodes for the agent to get the first positive reward due to the large state-action space.

We compare our method MCTS-DDU with recently proposed methods shown as follows. Moreover, we propose a stronger baseline called DDU.

- **DQN**: Agent is trained by DQN (Eq.1) using real experiences only.
- **Deep Dyna-Q (DDQ(K))**: Agent is trained by DQN (Eq.1) with background planning. The ratio between simulated experiences and real experiences is $K-1$ (Peng et al., 2018).
- **Switch-based Active Deep Dyna-Q (Switch-DDQ)**: Agent is trained by DQN (Eq.1) with background planning. The switcher automatically controls the ratio between simulated experiences and real experiences (Wu et al., 2018).
- **DDU**: Agent uses dueling architecture as Q-network and is trained by DDQN (Eq.2) using real experiences only.
- **MCTS-DDU** Agent is trained in the same way as **DDU**. While in the decision time, actions are picked based on MCTS with Q-network.

For all agents, the main components of Q-networks and models are implemented as two-layer neural networks with the hidden size being 80 and ReLU activation.

4.2 Evaluation with User Simulator

In this part, agents are evaluated by interacting with the same user simulator used in the training stage. However, a reserved part of the goal set is used for testing here. We evaluate the performances of each agent on the test goal set by the end of every training episode. The evaluation process runs 50 trials and averages the success rates. The evaluation results of all agents are summarized in Table 1. To align with the settings of Wu et al. (2018), we sample the success rates at episode 100, 200 and 300. MCTS-DDU has the highest performance at all times. Note that, MCTS-DDU continues to learn even after episode300. Detailed numerical comparisons are presented in the following parts.

DDQ(K) is an important group of baselines as the first proposed deep planning method in dialogue tasks, but its performances are reported with

Agent	Episodes100			Episodes200			Episodes300		
	Success	Return	Turns	Success	Return	Turns	Success	Return	Turns
DQN[2]	.2867	-17.35	25.51	.6733	32.48	18.64	.7667	46.87	12.27
DDQ(5)[1]	.6056	20.35	26.65	.7128	36.76	19.55	.7372	39.97	18.99
DDQ(5)[2]	.6200	25.42	19.96	.7733	45.45	16.69	.7467	43.22	14.76
DDQ(5)[3]	.6456	28.48	21.83	.6394	29.96	17.28	.6344	28.34	18.92
DDQ(10)[1]	.6624	28.18	24.62	.7664	42.46	21.01	.7840	45.11	19.94
DDQ(10)[2]	.6800	34.42	16.36	.6000	24.20	17.60	.3733	-2.11	15.81
DDQ(10)[3]	.6254	25.71	22.59	.6759	31.99	19.61	.7209	39.24	17.92
DDQ(20)[2]	.3333	-13.88	29.76	.4467	5.39	18.41	.3800	-1.75	16.69
DDQ(20)[3]	.7076	45.73	16.15	.8182	51.33	16.15	.7968	48.37	15.65
Switch-DDQ[2]	.5200	15.48	15.84	.8533	56.63	13.53	.7800	48.49	12.21
DDU	.4675	14.15	24.01	.7611	33.89	17.41	.8562	43.07	15.69
MCTS-DDU	**.7312**	46.63	19.77	**.9090**	57.26	12.79	**.9314**	55.87	12.13

Table 1: The performance summary of MCTS-DDU and baselines in terms of success rate and discounted cumulative return. (K) stands for K planning steps. MCTS-DDU uses $c = 4$ in UCT_D and runs 50 simulations per dialogue turn. The results are sampled at Episode 100, 200, and 300 and are averaged over three random seeds. All model parameters are initialized randomly without extra human conversational data pre-training. Superscripts indicate the data sources, [1] for (Peng et al., 2018), [2] for (Wu et al., 2018), and [3] for our own implementations based on open-sourced codes.

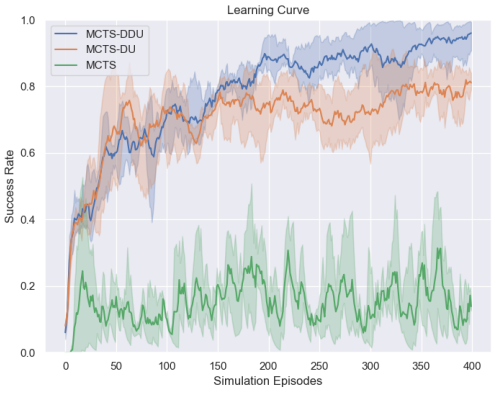

Figure 6: Ablation of advantages function.

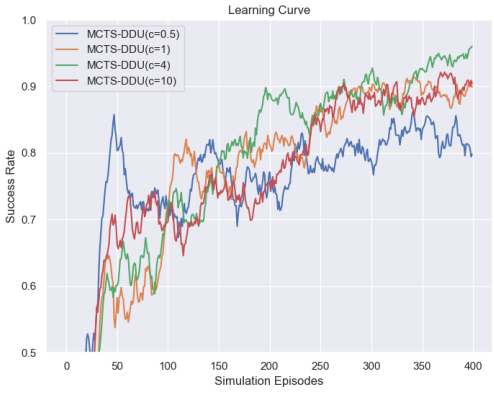

Figure 7: Effects of the balancing coefficient c.

large differences in Peng et al. (2018) and Wu et al. (2018). We reproduce it on our own and present the results in Table 1 and Figure 4. We then study the effectness of planning step K and select the best K that results in the highest average success rate in long term as the representative of the group DDQ(K) for the latter studies. The learning curves of $K = (2, 5, 10, 20)$ are shown in Figure 4. We have the similar results to Peng et al. (2018) that larger values of K make the learning process faster and success rate higher.

Then we compare DQN, DDQ(20), and MCTS-DDU. The result is shown in Figure 5. Methods incorporated with planning outperform than DQN both in training efficiency and success rate significantly, which proves the effectiveness of planning. MCTS-DDU achieves the highest task performance and data efficiency. Technically, MCTS-DDU exceeds DDQ(20) by absolute 12.65% and relative 14.79%. And it exceeds DQN by absolute 28.77% and relative 41.44%.

We define the number of episodes taken for achieving 60% success rate as a metric for comparing training efficiency. With this setting, MCTS-DDU is relatively 45.31% faster than DDQ(20) and 78.26% faster than DQN. Moreover, MCTS-DDU can reach over 50% success rate within 10 episodes.

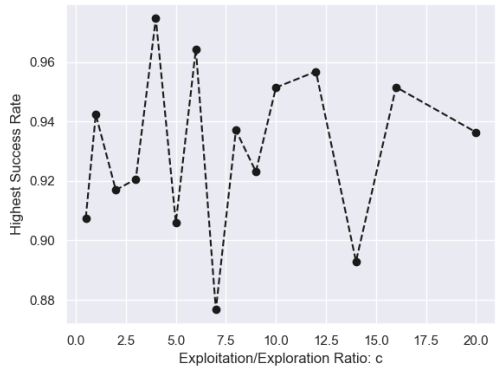

Figure 8: The highest success rates under different c.

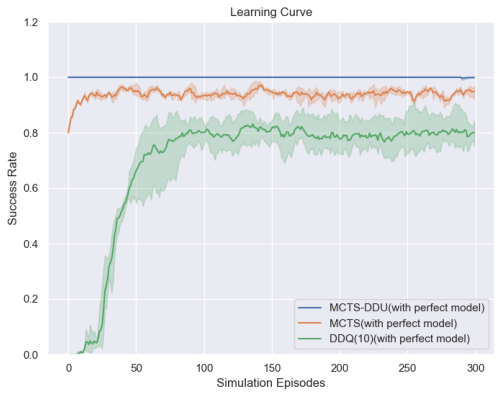

Figure 9: Performance comparison between background /decision-time planning with a perfect model.

4.3 Ablation studies

Effectiveness of MCTS We compare the performances between MCTS-DDU and DDU that exploits Q-network directly. The result is shown in the Table 1 and Figure 5. Based on the metric defined above, MCTS-DDU exceeds DDU by absolute 8.9% and relative 9.8% with 73.3% faster efficiency.

One interesting observation is that DDU could achieve slightly higher performance than DDQ(20) in spite of the lower training efficiency. It shows that the direct reinforcement learning part has not been investigated enough in the dialogue policy learning scenario. Solely using the advanced value-based learning methods can actually bring considerable improvement. Thus, we consider DDU to be a stronger baseline model than DQN for future study, based on which more complex mechanisms like planning can be added on.

Effectiveness of advantages and double-q Next, we investigate the effectiveness of the advantages function $A(s, a; \phi)$ and DDQN objective incorporated in UCT$_D$. The result is shown in the Figure 6. Without advantages as prior knowledge for exploration, the performance of plain MCTS is much worse than that of MCTS-DDU. The success rate fluctuates drastically due to the fact that more simulations are needed to make the action value estimations converged. We observe a slow and unstable learning process, implying merely using $V(s; \theta)$ is insufficient. Therefore, the conclusion can be reached that the advantages function $A(s, a; \phi)$ indeed improves the efficiency of simulations and is also critical for the high performance guarantee. We also perform the experiment in which the DDQN objective(Eq. 2) is replaced with the original DQN objective(Eq. 1). Even though the learning processes are quite commensurate in the early stage, MCTS-DU runs into a performance bottleneck after 200 episodes.

Exploitation v.s. Exploration We also explore how the coefficient c, balancing exploitation and exploration in UCT$_D$, effects task performances. We set the testing range to be roughly from 2^{-1} to 2^4 and the results are shown in Figure 7 and 8. As c increases from 0.5 to 4, the final success rate gets improved, emphasizing the importance of exploration. However, the performance starts to degenerate when c continues to increase. But it is still higher than those of the cases where c is small. Empirically, we believe a well-guided exploration, such as guided by an advantage function, is more influential in this task. In short, this experiment result is a clear illustration of the tradeoff between exploitation and exploration when using MCTS.

From a practical perspective, the coefficient c needs to be searched carefully for the optimal value. We present the highest success rates under different settings of c within 400 episodes in Figure 8.

Performance upper bound comparisons Lastly, we compare the performances of DDQ(10) and MCTS-DDU under perfect modeling learning. By "perfect", we mean there is no error in learned transition and reward functions. Our goal is to investigate the performance upper bounds of background and decision-time planning. The result in Figure 9 shows that DDQ(10) still stucks in a local optimal whereas MCTS-DDU can solve the task perfectly. We argue that the bottleneck of DDQ(10) comes from the use of a value function approximator. Contrastly, decision-time planning

is able to build a true sub-MDP with a perfect model and solves it exactly. In addition, we test the plain MCTS in this setting. It almost perfectly solves the task but is less stable than MCTS-DDU, which again demonstrates the effectiveness of the advantage function.

5 Conclusions

Our work introduces a novel way to apply deep model-based RL to task-completion dialogue policy learning. We combine the advanced value-based methods with MCTS as decision-time planning. In the movie-ticket booking task, MCTS-DDU agent exceeds recent background planning approaches by a wide margin with extraordinary data efficiency.

In this paper, one main focus is to demonstrate the differences between background and decision-time planning. However, it is reasonable and straightforward to combine them together. This might be an interesting topic for future work.

Acknowledgments

We thank Xuan Li and Haiqin Yang for their thoughtful comments on the paper.

References

Leemon C Baird III. 1993. Advantage updating. Technical report, WRIGHT LAB WRIGHT-PATTERSON AFB OH.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-carlo tree search: A new framework for game ai. In *AIIDE*.

Hado V Hasselt. 2010. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621.

He He, Derek Chen, Anusha Balakrishnan, and Percy Liang. 2018. Decoupling strategy and generation in negotiation dialogues. *arXiv preprint arXiv:1808.09637*.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Levente Kocsis, Csaba Szepesvári, and Jan Willemson. 2006. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1.

Xiujun Li, Zachary C. Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. 2016. A user simulator for task-completion dialogues.

Long Ji Lin. 1991. Programming robots using reinforcement learning and teaching. In *AAAI*, pages 781–786.

Zachary Lipton, Xiujun Li, Jianfeng Gao, Lihong Li, Faisal Ahmed, and Li Deng. 2018. Bbq-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, Kam-Fai Wong, and Shang-Yu Su. 2018. Deep dyna-q: Integrating planning for task-completion dialogue policy learning.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 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*, pages 149–152. Association for Computational Linguistics.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

S. Singh, D. Litman, M. Kearns, and M. Walker. 2002. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research*, 16:105–133.

Shang-Yu Su, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Yun-Nung Chen. 2018. Discriminative deep dyna-q: Robust planning for dialogue policy learning.

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

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Richard S Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael P Bowling. 2012. Dyna-style planning with linear function approximation and prioritized sweeping. *arXiv preprint arXiv:1206.3285*.

Ryuichi Takanobu, Hanlin Zhu, and Minlie Huang. 2019. Guided dialog policy learning: Reward estimation for multi-domain task-oriented dialog. *arXiv preprint arXiv:1908.10719*.

Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning*, 8(3-4):279–292.

Yuexin Wu, Xiujun Li, Jingjing Liu, Jianfeng Gao, and Yiming Yang. 2018. Switch-based active deep dyna-q: Efficient adaptive planning for task-completion dialogue policy learning.

Tiancheng Zhao, Kaige Xie, and Maxine Eskenazi. 2019. Rethinking action spaces for reinforcement learning in end-to-end dialog agents with latent variable models. *arXiv preprint arXiv:1902.08858*.

A Appendices

A.1 User goal set

The user goal set consists of 280 goals from real human conversation in the movie domain. We split it into 70%, 15%, and 15% for training, validation(hyperparameter tuning), and testing respectively. A sample user goal is shown as follows, where $constraint\_slots$ are the determined parts of a user goal and $request\_slots$ are the slots that need to be recommended by the agent.

{

$constraint\_slots$:

moviename: star wars

#people: 2

state: illinois

city: du quoin

$request\_slots$:

date: ?

theater: ?

starttime: ?

}

The feasible action sets of the user simulator and the agent are defined by the schema of intent and slot.

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

Table 2: The schema of intent and slot.

A.2 Hyperparameters

The main hyperparameters used in our method are listed in Table 3.

Hyperparameter	Search range	Optimal value
Max dialogue turns (Horizon)	None	32
Replay buffer capacity	None	5000
Batch size	{16, 32, 64, 128}	16
Optimizer	{SGD, RMSprop, Adam, AdamW}	AdamW
Epochs	[1, 15] (Step=5)	10
Learning rate	[1e-5, 1e-3] (Step=5e-5)	5e-3
ϵ-greedy (ϵ)	[0.05, 0.3] (Step=0.05)	0.2
Discount (γ)	[0.4, 0.9] (Step=0.1)	0.5
Exploitation v.s. Exploration (c)	[1,20] (Step=1)	4

Table 3: Hyperparameters for MCTS-DDU. $\{\cdot\}$ indicates the exact value range. $[\cdot]$ indicates the lower and upper bound for searching. Hyperparameter tunings are based on task success rates and performed over three random seeds. We use one NVIDIA V100 GPU as computing infrastructure and average runtime is about 2 hours per trial.