Context-Uncertainty-Aware Chatbot Action Selection via Parameterized Auxiliary Reinforcement Learning

Chuandong Yin[1], Rui Zhang[1(✉)], Jianzhong Qi[1], Yu Sun[2], and Tenglun Tan[1]

[1] The University of Melbourne, Parkville, Australia
{chuandongy,tenglunt}@student.unimelb.edu.au,
{rui.zhang,jianzhong.qi}@unimelb.edu.au
[2] Twitter, Inc., San Francisco, California, USA
ysun@twitter.com

Abstract. We propose a context-uncertainty-aware chatbot and a reinforcement learning (RL) model to train the chatbot. The proposed model is named *Parameterized Auxiliary Asynchronous Advantage Actor Critic* (PA4C). We utilize a user simulator to simulate the uncertainty of users' utterance based on real data. Our PA4C model interacts with simulated users to gradually adapt to different users' utterance confidence in a conversation context. Compared with naive rule-based approaches, our chatbot trained via the PA4C model avoids hand-crafted action selection and is more robust to user utterance variance. The PA4C model optimizes conventional RL models with action parameterization and auxiliary tasks for chatbot training, which address the problems of a large action space and zero-reward states. We evaluate the PA4C model over training a chatbot for calendar event creation tasks. Experimental results show that our model outperforms the state-of-the-art RL models in terms of success rate, dialogue length, and episode reward.

1 Introduction

Recently, personal assistants [11, 10, 12, 14] become increasingly popular, such as Apple Siri, which can interact with human and provide with intelligent service. These personal assistants are also called task-oriented chatbots ("*chatbot*" for short in the rest of the paper) that help users complete tasks of certain domains, such as creating a calendar event. A task may consist of several *slots*, such as *time* and *location* for a calendar event. A chatbot needs to identify these slots correctly via a dialogue with a user. Due to the uncertainty of language fluency among different people, it is not a simple task to identify these slots accurately. This problem is particularly challenging if the user utterance confidence given by *automatic speech recognition* (ASR) and *natural language understanding* (NLU) systems is low (e.g., a user is new to the language spoken or has a heavy accent).

A straightforward approach to tackle this problem is to confirm with a user whenever his or her utterance confidence is lower than a predefined threshold. However, in a practical scenario, this threshold may vary from person to person,

Table 1: Example dialogues of a rule-based chatbot and a context-uncertainty-aware chatbot.

Turn	**Role**	**Rule-based chatbot**	**Context-uncertainty-aware chatbot**
1	user	*inform* (title=[(“dinner”, 0.6)], invitee=[(“Mike”, 0.6)])	*inform* (title=[(“dinner”, 0.6)], invitee=[(“Mike”, 0.6)])
2	bot	*confirm* (title)	*confirm* (title)
3	user	*inform* (title=[(“dinner”, 0.7)])	*inform* (title=[(“dinner”, 0.7)])
4	bot	*confirm* (invitee)	*request* (time)
5	user	*inform* (invitee=[(“Mike”, 0.7)])	*inform* (time=[(“6 p.m.”, 0.6)])
6	bot	*request* (time)	*request* (location)
7	user	*inform* (time=[(“6 p.m.”, 0.6)])	*inform* (location=[(“Korean BBQ”, 0.6)])
8	bot	*confirm* (time)	*complete* ()
9	user	*inform* (time=[(“6 p.m.”, 0.7)])	
10	bot	*request* (location)	
11	user	*inform* (location=[(“Korean BBQ”, 0.6)])	
12	bot	*confirm* (location)	
12	user	*inform* (title=[(“Korean BBQ”, 0.7)])	
14	bot	*complete* ()	

and is also related to a specific dialogue context. Meanwhile, users would expect chatbots not only to complete the tasks required, but also complete in a limited *dialogue length*. A fixed threshold cannot adapt to various users' utterance confidences and may lead to lengthy dialogues, which might discourage the use of chatbots. To illustrate the problem, We use Table 1 to show two interaction sequences between a user and two different chatbots to create a calendar event for “dinner with Mike at 6 p.m. at Korean BBQ”. The user input in these sequences is represented as an *“inform”* tuple, which contains slots including *title*, *invitee*, *time*, and *location* of the event. These slots are generated by ASR and NLU systems, which are beyond the scope of our study. Each slot is associated with a number representing the user utterance confidence proposed by the ASR and NLU systems. The “Rule-based chatbot” column showcases how a rule-based chatbot may interact with the user. In each turn, the rule-based chatbot takes one of three possible actions as a response to user input: (1) *confirm* (to request a confirmation of a slot previously captured from user input with low confidence), (2) *request* (to request a new slot from user), and (3) *complete* (to set up the calendar event and finish the conversation). The rule-based chatbot confirms with the user for each slot until its confidence reaches a fixed threshold 0.7. This has resulted in a lengthy dialogue with 14 turns.

We aim to overcome the problem of fixed confidence thresholds as illustrated above with a chatbot, which can adaptively choose a threshold according to the user and dialogue context. We call such a chatbot a *context-uncertainty-aware* chatbot. The “Context-uncertainty-aware chatbot” column of Table 1 illustrates how such a chatbot will interact with a user. This chatbot also has a starting confidence threshold of 0.7, and it needs to confirm with the user for the first slot (*title*) that has a confidence below this threshold (Turn 2). Once this is confirmed, the chatbot learns that only a threshold of 0.6 is sufficient to accept the input of this user. As a result, the *invitee* slot (and any slots afterwards)

which also has a confidence of 0.6 does not need a confirmation anymore. This has shortened the dialogue to 8 turns and improved the user experience.

We take the above issues into account and propose an RL model named *PA4C* for chatbot training. This model addresses the following two problems of existing RL models. The first problem is that, in chatbot training, traditional RL models often have a large space for action selection, making it difficult to learn the best action to be selected with a large reward. The output of these models at a turn of the chatbot is a one-hot vector indicating which action should be selected. An action of the chatbot consists of two components: a function (the type of actions) and its parameter (slots). For example, the action *request(time)* has a function *request* and a parameter *time*. Traditional RL models simply list all possible combinations of functions and slots. Suppose that there are M action functions and N slots. Then the number of actions in these models will be $M \times N$. To reduce the action space and improve the reward, we introduce the *action parameterization* to separate the actions into two channels: one for functions and the other for parameters (cf. Fig 1b). In this way, the action space can be reduced from quadratic (i.e., $M \times N$) to linear (i.e., $M + N$).

The second problem addressed is that only a few states in dialogues have positive rewards. This makes early discovery of states that may lead to large rewards difficult. RL models thus may encounter a bottleneck due to missing valuable states. Traditional methods only focus on the target task (e.g., predict the target action) without explicitly paying attention to estimate the reward of states. Inspired by [2], we propose to add additional tasks to the chatbot during training and guide it to discover large-reward states (detailed in Section 5.2). In particular, we design two auxiliary networks: (1) a reward prediction network for predicting the reward of dialogue states, and (2) a value function replay network for helping the RL model estimate the expected state value.

This paper makes the following contributions:

- To the best of our knowledge, we are the first to propose a context-uncertainty-aware chatbot that is self-adaptive to the uncertainty of users' utterance confidence in a dialogue context via reinforcement learning.
- To overcome the quadratic action space problem in chatbot training by reinforcement learning, we propose the action parameterization technique which learns the functions and slots in two separate channels.
- We further propose two auxiliary networks to guide our model to pay extra attention to valuable states, which is more robust to both short-term immediate rewards and long-term expected returns.

2 Related Work

Most commercial chatbots are based on hand-crafted rules design for dialogue state tracking and action selection, which naively choose the action with the largest NLU confidence [16]. It is non-trivial to create a large set of rules to cover diverse user utterances. To avoid manually developing rules, machine learning approaches have been used to build chatbots. Chatbot training has been modeled

as a *Markov decision process* (MDP) [3] or a *partially observable Markov decision process* (POMDP) [17]. It is then formulated as a sequential labeling problem in *The Dialog State Tracking Challenge* (DSTC) [16, 1].

Machine learning approaches, however, need a large amount of training data, and it is labor-intensive to prepare such data. *Reinforcement learning* (RL) then is used to reduce the amount of training data needed. A noticeable progress has been made on training chatbots with RL models [18, 4, 8], which demonstrates the feasibility of training chatbots via reinforcement learning. However, due to the "cold-start" problem in reinforcement learning, existing RL approaches have to use supervised learning as a bootstrap. These approaches may interfere with the action exploration of reinforcement learning and cause many ceilings on the success rate, length and episode reward of dialogues. To the best of our knowledge, no existing studies have tackled the problem of breaking the ceilings of RL models in chatbot training. Our study aims to address it.

3 Preliminaries

We start with basic concepts in *deep reinforcement learning* (DRL). A DRL model is essentially a *Markov decision process* (MDP). It can be defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ denotes the environment state; $\mathcal{A}$ denotes the action space; $\mathcal{P}$ denotes the transition probability $P(s_{t+1}|s_t, a_t)$; $\mathcal{R}$ denotes the expected immediate reward function $\mathcal{R}(s_t, a_t)$; γ denotes a discount factor, $\gamma \in (0, 1]$ [19]. The goal of a DRL model is to maximize the *return* (cumulative expected rewards from the state s_t) $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, where $r_t = \mathcal{R}(s_t, a_t)$. To maximize the return, there are two approaches in general: value-based DRL and policy-based DRL.

3.1 Value-based DRL

Value-based DRL estimates the value of executing different actions in a state, and selects the action with the largest value. A typical value-based model is *Q-learning*, where "Q" represents the action value. Q-learning defines an action-value function $Q^{\pi}(s, a)$, where a is an action, s is a state, and π is a policy to be learned. It aims to find an optimal policy π with the maximum Q value according to $Q^*(s, a) = \max_{\pi} \mathbb{E}[G_t|s_t = s, a_t = a, \pi]$ and $\pi^*(s) = \text{argmax}_a Q^*(s, a)$ [6]. *Bellman equation* [15] is used to search for the optimal policy:

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}} \left[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) | s_t, a_t \right] \tag{1}$$

Here, s_{t+1} is the next state and a_{t+1} is any possible actions for s_{t+1}.

3.2 Policy-based DRL

Instead of directly optimizing the policy, value-based DRL models estimate a Q value for each action and chooses the action with the maximum Q value.

Table 2: Available actions and slots in the user simulator.

User Actions	*inform(slot), confirm_deny(slot), confirm_accept(slot), complete(), abort(), dont_care(slot)*
System Actions	*greeting(), request(slot), confirm(slot), complete(), abort()*
Slots	title, time, invitee, location

Table 3: User intent database: the number in each tuple represents a confidence level, which is the product of ASR and NLU confidence.

id	title	time	location	invitee
1	("dinner", 0.9)	("7 p.m.", 0.95)	("Korean BBQ", 0.87)	("Michael", 0.54), ("Mike", 0.46)

This approach may cause some biases. For example, assume that there are two actions a_1 and a_2 with Q values 50 and 49, respectively. The action a_2 will not be selected, although it may have a larger long-term value than that of a_1.

Policy-based DRL models are proposed to tackle this problem by directly optimizing the policy [13]. A typical approach is called *policy gradient*, which can define a stochastic policy $a = \pi(a|s;u)$, where u is the weight. The total reward can be computed as:

$$L(u) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + ...|\pi(\cdot;u)\right] \tag{2}$$

The gradients can be updated via [13]:

$$\frac{\partial L(u)}{\partial u} = \mathbb{E}\Big[\frac{\partial log\pi(a|s;u)}{\partial u} Q^{\pi}(s,a)\Big] \tag{3}$$

4 User Simulator

We construct a user simulator for chatbot training. As an example application, we focus on training a chatbot for calendar event creation tasks, although the techniques proposed may be applied to train a chatbot for other tasks. All available user actions, system actions, and slots are defined in Table 2.

4.1 Data Preparation

We collect 300 calendar events from volunteers through a data collector website, including 300 titles, 300 time, 113 invitees and 173 locations. Since the uncertainty produced by ASR and NLU may cause a user's real intent to be distorted, such as "Mike" being misunderstood into "Michael", we construct a database *IntentDB* to store these intents with probabilities as shown in Table 3. In addition, we calculate the mean μ and the standard deviation σ of the ASR & NLU uncertainty for each dialogue in The Dialogue State Tracking Challenge 2 (DSTC2) dataset (in the domain of restaurant booking) [16]. We save the pairs of μ and σ into a *NoiseDB*. At each step of a dialogue simulation, a Gaussian noise with μ and σ will be introduced to augment the collected data (detailed in Section 4.2). In this way, we can simulate millions of dialogues by adding different noises to the 300 events collected from volunteers.

Algorithm 1 User simulation

Initialize:
  $T \leftarrow 0,\ complete \leftarrow False,\ terminal \leftarrow False$
  $\Omega \leftarrow RandomSelect(IntentDB)$ // user intent (a distribution)
  $\omega \leftarrow Sample(\Omega)$ // user real goal
  $\omega' \leftarrow \{\}$ // chatbot recorded goal
  $z \leftarrow RandomSelect(NoiseDB)$ // noise, a pair of (μ, σ)
  $\mathcal{A}_{user} \leftarrow null$ // user action
  $\mathcal{A}_{bot} \leftarrow Action.greeting$ // chatbot action
repeat
  $\mathcal{A}_{user} \leftarrow UserRespond(\mathcal{A}_{bot}, \Omega, z)$ // randomly selected with $\mathcal{A}_{bot}, \Omega, z$
  $\mathcal{A}_{bot} \leftarrow ChatbotRespond(\mathcal{A}_{user})$ // ouputted by RL model
  $T \leftarrow T + 2$
  $\omega' \leftarrow \omega' + ParseEntity(\mathcal{A}_{bot})$
  if $T > T_{max}$ or $\mathcal{A}_{bot} == Action.complete$ **then**
    $terminal \leftarrow True$
  $reward \leftarrow R(terminal, complete, T)$
until $terminal$
if $\omega == \omega'$ **then** $complete \leftarrow True$
$reward \leftarrow R(terminal, complete, T)$

4.2 Dialogue Simulation

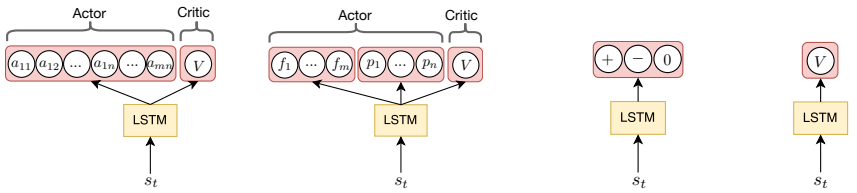
The simulation is based on the fact that a user's intent is known by himself (i.e., user simulator) but unknown by the chatbot. A step of a simulation includes two parts: (1) the chatbot asks the user a question, and (2) the user answers the question honestly. Each step will be assigned with a scalar reward according to the reward function $\mathcal{R}(terminal, complete, T)$ defined in the Eq. 4. The goal of a chatbot is to learn to ask a user questions which can maximize the total reward of a dialogue. The intuition of $\mathcal{R}$ is to give a higher reward to states that lead to a short dialogue that completes a user task successfully.

$$\mathcal{R}(terminal, complete, T) = \begin{cases} -0.1, & \text{if not } terminal \\ 2.0 \times \dfrac{T_{max} - T}{T_{max}}, & \text{if } terminal \text{ and } complete \\ -1.0, & \text{if } terminal \text{ and not } complete \end{cases} \tag{4}$$

Here, *terminal* indicates whether the simulation finishes, *complete* indicates whether the chatbot completes the task, T represents the length of the current dialogue, and T_{max} represents the predefined maximum length of a dialogue. A simulation terminates once T_{max} steps are executed regardless whether the user task has been completed. The reward function is Markovian because it only depends on the current dialogue state (i.e. *terminal*, *complete*, and T). At the end of each simulation, we compare the user's real goal ω (i.e., the *ground truth*) with the entities ω' captured by the chatbot. If ω is equal to ω', it means that the chatbot completes the user's task. Otherwise, the task fails. Algorithm 1 shows the detail of a simulation.

5 Proposed Model

As mentioned in Section 2, existing works usually use supervised learning as bootstrap before training with RL models. This approach may interfere with the



(a) Vanilla A3C (b) Parameterized A3C (c) RP Task (d) VR Task

Fig. 1: Model structure: (a) Vanilla A3C model. (b) Parameterized A3C model. (c) Reward prediction sub-model. (d) Value function replay sub-model.

action exploration and cause a ceiling on RL models because the data collected may produce some biases. In this paper, we propose a model named *parameterized auxiliary asynchronous advantage actor-critic* (PA4C) based on the A3C model [5]. Our PA4C model consists of two parts: (1) **Parameterized A3C** (PA3C), which solves the huge action space problem in traditional RL models for chatbot training; (2) **Auxiliary tasks**, which helps model discover the states with large rewards and enhances model robustness [2].

5.1 Parameterized A3C

PA3C is built on the vanilla A3C model, which is a hybrid value-based and policy-based DRL model. It can be expressed by two separate networks with shared weights (cf. Fig 1a): (1) *Actor* network outputs the policy $\pi(a_t|s_t;u)$; (2) *Critic* network estimates a state value function $V(s_t;v)$ [5], where u and v denotes the weights of the actor and critic network, respectively.

$$V(s_t;v) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t\right] \tag{5}$$

A function $A(s_t, a_t; v, u)$ is defined in Eq 6 to estimate the advantage of executing a_t on s_t over the state value $V(s_t;v)$:

$$A(s_t, a_t; u, v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};v) - V(s_t;v) \tag{6}$$

Therefore, the weights u of actor network can be updated as follow:

$$\frac{\partial L(u)}{\partial u} = \frac{\partial log\pi(a_t|s_t;u)}{\partial u} A(s_t, a_t; u, v) \tag{7}$$

The critic network can be optimized by minimizing the MSE loss [5]:

$$L(v) = (A(s_t, a_t; u, v))^2 \tag{8}$$

Like other RL models, vanilla A3C still cannot avoid the large action space problem. For example, assume that there are a set of M *functions* $\{f_1, f_2, ..., f_M\}$ and N *parameters* $\{p_1, p_2, ..., p_N\}$ available (functions and parameters are system

actions and slots respectively in the user simulator). For vanilla A3C, the number of actions is $M \times N$, because its actor network simply outputs all combinations of functions and parameters (cf. Fig 1a).

To address this problem, we propose PA3C by introducing action parameterization into the vanilla A3C model. Rather than listing all combinations, our model will split the policy π of vanilla A3C into two sub-policies π_f and π_p, which directly learns functions and parameters, respectively (cf. Fig 1b). In this manner, the number of actions in our model can be reduced from $M \times N$ (quadratic) to $M + N$ (linear). Correspondingly, the loss of actor network in PA3C has a slight difference. We modify the loss defined in Eq. 7 to Eq. 9, where u_f and u_p denote the weights of π_f and π_p respectively, and $\mathbf{\Lambda}$ is a mask vector for indicating whether the function f_m has a parameter.

$$\begin{aligned} \frac{\partial L(u)}{\partial u} &= \frac{\partial L(u_f)}{\partial u_f} + \mathbf{\Lambda}\frac{\partial L(u_p)}{\partial u_p} \\ &= (\frac{\partial log\pi_f(f_t|s_t;u_f)}{\partial u_f} + \mathbf{\Lambda}\frac{\partial log\pi_p(p_t|s_t;u_p)}{\partial u_p})A(s_t, f_t, p_t; u_f, u_p, v) \end{aligned} \tag{9}$$

5.2 Auxiliary Tasks

A recent DRL model named *UNsupervised REinforcement and Auxiliary Learning* (UNREAL) [2] suggests that incorporating reasonable auxiliary tasks can improve the model robustness and performance. In chatbot training, the rewards are usually sparse. Only a few states can provide immediate rewards (e.g., large negative or positive rewards only occurs when the chatbot finishes the task). The rewards of most states during a conversation are zeros or very small numbers, making it difficult to learn the values of these states. This may cause many ceilings on RL models used in chatbot training, such as low success rate and lengthy dialogues. To address this problem, we design two auxiliary networks to assist the PA3C sub-model to take into account both short-term immediate rewards and long-term expected returns.

Reward Prediction (RP) is an auxiliary network to predict the immediate reward of the next unseen state given a historical context. It helps PA3C sub-model better evaluate the value of dialogue states, To train this task network, we sample a historical sequence $S_t = (s_{t-k}, s_{t-k+1}, ..., s_{t-1})$ from a replay buffer and predict the reward r_t of the state s_t. Here, we focus on whether the state is valuable rather than the specific reward. Instead of estimating the real value of r_t, RP only predicts the sign of r_t in three classes: *positive*, *negative* and *zero* (cf. Fig 1c).

Value function Replay (VR) is an auxiliary network to enhance the state-value function $V(s_t; v)$ (Eq. 5) of the critic network in the PA3C sub-model. The function $V(s_t; v)$ is designed to estimate the long-term expected return of the current state s_t. Therefore, VR shares weights with the critic network in PA3C. The only difference is that the critic network is trained with on-policy, while VR can be trained with off-policy. It will sample a state sequence as input from a replay buffer like the RP network (cf. Fig 1d). In this way, $V(s_t; v)$ combines the strength of both on-policy and off-policy training, which is more robust when estimating the expected return.

5.3 PA4C model

Our full PA4C model integrates the PA3C sub-model and auxiliary networks. The full model is illustrated in Fig 2. Firstly, the PA3C sub-model interacts with the user

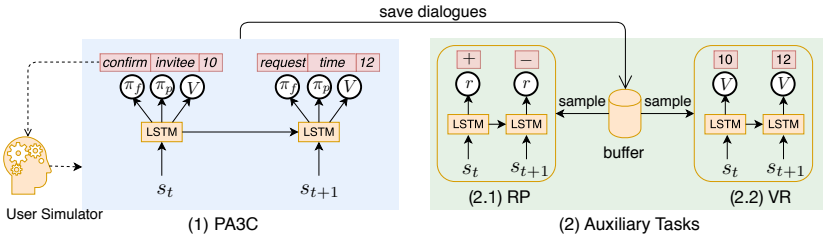


Fig. 2: Full PA4C model consists of (1) PA3C and (2) Auxiliary Tasks (RP and VR networks). All LSTM layers share the same weights.

simulator, which generates a dialogue. The dialogue will be saved into a small replay buffer, where RP and VR can sample historical sequences to update their weights. The final loss of PA4C is the combination of PA3C, RP, and VR networks [2]:

$$L_{PA4C} = L_{PA3C} + \lambda_{RP} L_{RP} + \lambda_{VR} L_{VR} \tag{10}$$

where λ_{RP} and λ_{VR} are the weight factors of the RP and VR networks respectively; L_{PA3C} is the loss of the PA3C network defined in Eq. 9; L_{RP} is the cross entropy loss of the RP network; L_{VR} is the MSE loss (cf. Eq. 8) of the VR network.

Our PA4C model also extends the asynchronous training from vanilla A3C via multiple threads. First, PA3C will create a global network in the main thread and multiple local networks in several independent threads. The global network is responsible for dispatching gradients to each local network, which will run a user simulator to compute the local gradients. At the end of each simulation, the gradients of local networks will be aggregated back to the global network.

6 Experiments

In this section, we compare the performance of chatbots trained with our PA4C model and with baseline models in six metrics: *success rate* (SR), *dialogue length* (DL), *episode reward* (ER), and their standard deviations ("std" for short in the rest of the paper) over 10 runs, i.e., std SR, std DL, and std ER [9, 7]. All the DRL models are trained on 180 dialogues with noises and evaluated on 120 dialogues. The result shows that our model outperforms the state-of-the-art models in these metrics for training a chatbot in the calendar event creation task.

6.1 Baseline Models and Hyperparameters

We implement 4 baselines, including a rule-based model and 3 existing DRL models:

– **Rule-based** We implement a rule-based chatbot by *if-else* triggers. If a slot is informed by a user and its confidence is larger than a predefined threshold 0.724 (the average confidence of the collected events), the chatbot will request the next slot; Otherwise, the chatbots will confirm it with the user. When all slots are obtained, the chatbot will finish a simulation.

- **DQN** We stack two consecutive states as the input s_t at step t, and use two fully-connected layers with 256 and 64 hidden units, respectively. The replay buffer size is 10^5 and batch size is 128. In the first 10^6 step, we use the *epsilon-greedy* policy to explore actions, with ε annealing from 1.0 to 0.1.
- **DRQN** We only feed one single state into DRQN at step t. Unlike DQN, the replay buffer for DRQN stores the full episode of a dialogue. The *timestep* of LSTM is set to 10. The remaining hyperparameters are the same as those in DQN.
- **A3C** We use LSTM in A3C, and update the gradients in local networks per episode rather than a fixed step in vanilla A3C [5]. We also set the regularization of policy entropy $\beta = 0.1$ to encourage action exploration and train with 16 threads.
- **A4C** We only add auxiliary networks into A3C. Both λ_{RP} and λ_{VR} in Eq. 10 are set to 1.0. The replay buffer for auxiliary networks is 2000. When training the RP network, we clip the rewards whose absolute value is smaller than 0.1 to 0 to get their sign (i.e., positive, negative, zero). We assume these small rewards cannot guide to discover valuable states. Other settings are identical to those in A3C.
- **PA3C** We only add action parameterization into A3C. The number of actions now becomes to 8. Other settings are identical to those in A3C.
- **PA4C** In our full PA4C model, we integrate PA3C with A4C sub-models. The hyperparameters of these two sub-models are the same as those in A4C and PA3C.

The input feature is composed of a 52-dimensional vector provided by the user simulator, including the last system action, the current user action, and the current dialogue length. All LSTM layers have 256 hidden units, followed by an ReLU activation function. The discounted factor of reward γ is 0.99 for all models. We use the RMSProp optimizer for gradients computing with learning rate $\eta = 0.001$ and weight decay $\alpha = 0.99$. The maximum dialogue length T_{max} is set to 20. The action *greeting()* is removed from the set of actions because *greeting()* is always called firstly by the chatbot. The number of actions for non-parameterized models and parameterized models are $M_p \times N + M_{np} = 2 \times 4 + 2 = 10$ and $M + N = 4 + 4 = 8$, respectively, where M_p denotes the number of functions with parameters, i.e., {*request(slot), confirm(slot)*}; M_{np} denotes the number of functions without parameters, i.e., {*complete(), abort()*}; N denotes the number of slots, i.e., {*title, time, location, invitee*}; $M = M_p + M_{np}$.

6.2 Results

In the section, we compare the performance of the PA4C model with baselines.

Comparison with the Rule-based Model: The result in Table 4 shows that PA4C outperforms the rule-based model in all metrics: 47% in success rate (SR), 52% in std SR, 42% in dialogue length (DL), 62% in std DL, 35% in episode reward (ER) and 54% in std ER. In particular, PA4C can achieve over 0.93 in SR, with less than 7.6 turns to complete the task. In contrast, the rule-based model just reaches 0.63 in SR while it takes more than 12 turns. This shows that PA4C can adapt to the utterance uncertainty according to the dialogue context. It can produce a high success rate and a short dialogue.

Comparison with RL Models: Table 4 also shows that PA4C outperforms existing RL models, with 14%, 19%, and 12% improvement in SR, DL, and ER, respectively, compared with the most recent RL model A3C. We further illustrate the learning curves of SR, DL, and ER in Fig 3. As the figure shows, although DQN has shorter dialogues, it sacrifices the success rate. It often fails when interacting with low-confidence users, while our PA4C model is more robust to these users.

Table 4: Performance comparison. Larger *success rate (SR)* and *episode reward (ER)* indicate better performance. Smaller *dialogue length (DL)*, *std SR*, *std DL* and *std ER* indicate better performance. The number in the parenthesis shows the improvement over the rule-based model. When computing the improvement, we scale the value of *ER* to $[0, +\infty]$ because of negative rewards. The results are averaged over 10 runs.

Model	SR	std SR	DL	std DL	ER	std ER
RULE	0.634	0.052	12.925	5.769	-0.286	1.018
DQN	0.794(+25%)	0.041(−21%)	**7.075**(−45%)	**1.889**(−67%)	0.443(+29%)	0.651(−36%)
DRQN	0.691(+09%)	0.046(−12%)	10.925(−15%)	2.429(−58%)	-0.194(+04%)	0.999 (−02%)
A3C	0.843(+33%)	0.037(−29%)	9.889(−23%)	3.075(−47%)	0.302(+23%)	0.560(−45%)
A4C	0.854(+35%)	0.034(−35%)	10.402(−20%)	3.675(−36%)	0.261(+22%)	0.669(−34%)
PA3C	0.900(+42%)	0.029(−44%)	8.382(−35%)	2.894(−50%)	0.519(+32%)	0.492(−52%)
PA4C	**0.932**(+47%)	**0.025**(−52%)	7.558(−42%)	2.216(−62%)	**0.585**(+35%)	**0.469**(−54%)

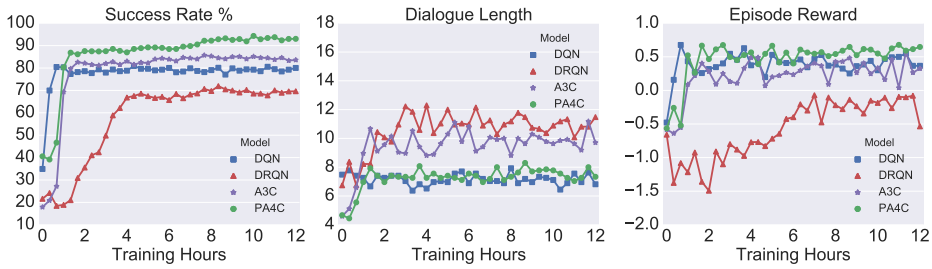


Fig. 3: Performance comparison with RL models: DQN, DRQN, A3C, and PA4C. Each figure is averaged over 10 runs.

Comparison within Sub-models: To verify the effect of action parameterization and auxiliary tasks, we compare A3C, A4C, PA3C, and PA4C. Table 4 shows that PA4C outperforms sub-models in all metrics. Using action parameterization achieves 9% improvement in SR, 12% in DL, 9% in ER over A3C. Although only adding auxiliary tasks to A3C (i.e., A4C) does not have such significant effect, the auxiliary tasks do help boost the model performance when integrated together with action parameterization to A3C (i.e., PA4C).

7 Conclusion

We presented a context-uncertainty-aware chatbot trained via reinforcement learning. A user simulator is designed to simulate the uncertainty of different users' utterance confidence. Our chatbot trained with this simulator can adapt to different users' utterance confidence based on the dialogue context. We proposed a reinforcement learning model named PA4C to optimize chatbot training, which can avoid a large action selection space via action parameterization and can discover valuable states via auxiliary tasks. We evaluate our model by training a chatbot for the calendar events creation task. Experimental results show that our PA4C model outperforms the state-of-the-art models in the metrics of success rate, dialogue length, and episode reward.

Acknowledgements This work is supported by Australian Research Council (ARC) Future Fellowships Project FT120100832 and Discovery Project DP180102050.

References

1. Henderson, M., Thomson, B., Young, S.: Deep neural network approach for the dialog state tracking challenge. In: SIGDIAL (2013)
2. Jaderberg, M., Mnih, V., Czarnecki, W.M., Schaul, T., Leibo, J.Z., Silver, D., Kavukcuoglu, K.: Reinforcement learning with unsupervised auxiliary tasks. ICLR (2017)
3. Levin, E., Pieraccini, R., Eckert, W.: Learning dialogue strategies within the markov decision process framework. In: Automatic Speech Recognition and Understanding, IEEE (1997)
4. Li, X., Chen, Y.N., Li, L., Gao, J.: End-to-end task-completion neural dialogue systems. arXiv preprint arXiv:1703.01008 (2017)
5. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: ICML (2016)
6. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature (2015)
7. Pietquin, O., Hastie, H.: A survey on metrics for the evaluation of user simulations. The knowledge engineering review (2013)
8. Rojas-Barahona, L.M., Gasic, M., Mrksic, N., Su, P., Ultes, S., Wen, T., Young, S.J., Vandyke, D.: A network-based end-to-end trainable task-oriented dialogue system. In: EACL (2017)
9. Schatzmann, J., Weilhammer, K., Stuttle, M., Young, S.: A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. The knowledge engineering review (2006)
10. Sun, Y., Yuan, N.J., Wang, Y., Xie, X., McDonald, K., Zhang, R.: Contextual intent tracking for personal assistants. In: SIGKDD (2016)
11. Sun, Y., Yuan, N.J., Xie, X., McDonald, K., Zhang, R.: Collaborative nowcasting for contextual recommendation. In: WWW (2016)
12. Sun, Y., Yuan, N.J., Xie, X., McDonald, K., Zhang, R.: Collaborative intent prediction with real-time contextual data. ACM TOIS (2017)
13. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: NIPS (2000)
14. Wang, Y., Yuan, N.J., Sun, Y., Qin, C., Xie, X.: App download forecasting: An evolutionary hierarchical competition approach. In: IJCAI (2017)
15. Watkins, C.J., Dayan, P.: Q-learning. Machine learning (1992)
16. Williams, J., Raux, A., Ramachandran, D., Black, A.: The dialog state tracking challenge. In: SIGDIAL (2013)
17. Williams, J.D., Young, S.: Partially observable markov decision processes for spoken dialog systems. Computer Speech & Language (2007)
18. Williams, J.D., Zweig, G.: End-to-end lstm-based dialog control optimized with supervised and reinforcement learning. arXiv preprint arXiv:1606.01269 (2016)
19. Zhao, T., Eskénazi, M.: Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. In: SIGDIAL (2016)