The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

CoChat: Enabling Bot and Human Collaboration for Task Completion

Xufang Luo,[†*] Zijia Lin,[‡] Yunhong Wang,[†] Zaiqing Nie[§]
[†]Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University, Beijing, China
[‡]Microsoft Research, Beijing, China
[§]Alibaba AI Labs, Beijing, China
{luoxufang,yhwang}@buaa.edu.cn; zijlin@microsoft.com; zaiqing.nzq@alibaba-inc.com

Abstract

Chatbots have drawn significant attention of late in both industry and academia. For most task completion bots in the industry, human intervention is the only means of avoiding mistakes in complex real-world cases. However, to the best of our knowledge, there is no existing research work modeling the collaboration between task completion bots and human workers. In this paper, we introduce CoChat, a dialog management framework to enable effective collaboration between bots and human workers. In CoChat, human workers can introduce new actions at any time to handle previously unseen cases. We propose a memory-enhanced hierarchical RNN (MemHRNN) to handle the one-shot learning challenges caused by instantly introducing new actions in CoChat. Extensive experiments on real-world datasets well demonstrate that CoChat can relieve most of the human workers' workload, and get better user satisfaction rates comparing to other state-of-the-art frameworks.

Introduction

Task completion bots are attracting lots of attention from both industry and academia. They aim to help users complete specific tasks (e.g., booking movie tickets) via more natural interactions, i.e., conversations. The focus of task completion bots is to complete tasks successfully, while achieving high user satisfaction. A good dialog manager is a crucial component for such task completion bots. It takes language understanding results as input, and decides which action (e.g., asking for a movie name) should be taken. Therefore, the dialog manager directly controls dialog flow, decides the success/failure of task completion, and also affects user satisfaction (Lee et al. 2010; Young et al. 2013).

Although there are many methods for developing an automatic dialog manager, such automatic systems are potentially problematic, as they don't share the same experience from human workers on how to avoid serious mistakes that would negatively affect users, e.g., failing the task or annoying them. Currently, *human intervention* is the only means of avoiding such mistakes in complex real-world domains (Saunders et al. 2017). Besides, human workers are already heavily involved in many common task completion systems, e.g., call centers. Hence, modeling the collaboration between bots and human workers is a reasonable approach. It can even be the *only* practical way for learning and using the dialog manager in many real-world applications.

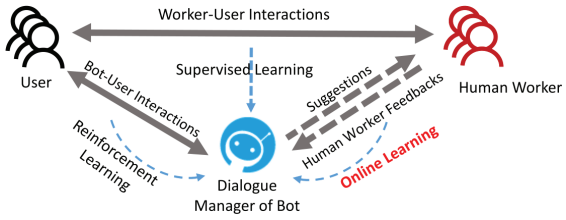


Figure 1: Illustration of the CoChat Framework

In this paper, we propose a practical dialog management framework named **CoChat** to enable effective collaboration between bots and human workers. To the best of our knowledge, our work is the first to model the collaboration between task completion bots and human workers, which is necessary and reasonable for many real-world applications. In CoChat, human workers can intervene in the learning process at any time, and the dialog manager can be continuously improved via learning from labeled dialog logs, human workers' feedback and users' feedback. As illustrated in Fig. 1, the dialog manager is firstly initialized via supervised learning. After that, the bot can collaborate with human workers, suggest actions for them to reduce their workload, and continuously learn from their feedback via online learning. When human workers are unavailable, e.g., off work, and users are willing to try the bot, the bot can directly interact with users, and its dialog manager can be further improved via reinforcement learning. Briefly, the dialog manager is continuously improved to maximize user satisfaction, and relieve human workers' workload during collaboration.

Having human workers involved, generally allows the introduction of new actions to handle complex or unseen cases, and thus raises challenges of learning new actions for the dialog manager. In previous works like (Lee et al. 2010; Su et al. 2017), the dialog manager requires a fixed action set, and then learns to select one action from it to handle each case. However, this setting is unrealistic when human workers are involved to handle complex or unseen cases, because

[*]This work was done when the first author was on an internship with Microsoft Research.

new actions outside the original action set will probably be required urgently to facilitate task completion. For example, a customer support center may receive questions about problems in their latest product, which may have never been seen before and therefore cannot be handled by the bot. Then "tell users how to handle these new problems" becomes a new required action, and it will be provided by human workers who intervene to handle such cases. The dialog manager should also learn this new action for better suggesting actions and handling similar cases later. Such challenges of learning new actions have not been well studied before. Though previous work can adapt to new actions by retraining their corresponding models, they suffer the risk of losing accumulated learned knowledge, as learning processes like reinforcement learning often cannot be reproduced exactly.

To implement the proposed CoChat framework and better handle the challenges of learning new actions, we further propose a novel dialog manager model termed MemHRNN. The proposed MemHRNN consists of a hierarchical recurrent neural network (i.e., HRNN) (Li, Luong, and Dan 2015; Xie et al. 2016; Yang; and Huang 2016; Sordoni et al. 2015) and an external memory. The HRNN combines dialog history, language understanding results and external information like API call results as input, and then outputs probability distributions over all actions for action selection. When new actions come up, the architecture of the HRNN can be changed accordingly without losing the knowledge accumulated in previous learning processes. Most importantly, the external memory in MemHRNN is further introduced to handle the one-shot learning challenges (Santoro et al. 2016; Graves, Wayne, and Danihelka 2014) caused by instantly introducing new actions, i.e., their corresponding occurrences are too few for the HRNN to learn valid strategies regarding them. Specifically, we record the occurrences of new actions in the memory, and leverage them to derive the probabilities of selecting these new actions, which are then merged with those output by the HRNN to enhance action selection.

We conduct experiments on two realistic tasks: i) booking restaurants; and ii) booking movie tickets. Experimental results show that: 1) In CoChat, when human workers are available, high user satisfaction can be achieved. Moreover, by suggesting actions for both tasks, the dialog manager learned by MemHRNN can reduce workers' workload by 91.35% and 86.32% respectively; 2) The learned dialog manager can achieve high user satisfaction rates, 97.04% and 92.62% respectively, of those achieved by human workers after continuous learning. 3) When new actions come up, MemHRNN can quickly learn strategies to cope with them via smooth architecture adaptations and leveraging the external memory.

The contributions of our work are summarized as follows:

- We propose CoChat, a practical dialog manager learning framework that models the collaboration between bots and human workers for task completion;
- We propose a memory-enhanced hierarchical RNN model termed MemHRNN to implement the CoChat framework and handle the one-shot learning challenges caused by instantly introducing new actions.

Related Work

In task completion bots, a dialog manager decides what the next action should be, given the current conversation state (Young 2006; Zhao and Eskenazi 2016). Generally, the conversation state is derived from message history, previous actions, natural language understanding (NLU) results, etc.. Different learning-based approaches, apart from handcrafted rules, have been proposed to develop such dialog managers.

Inspired by non-task-oriented bots (Vinyals and Le 2015; Shang, Lu, and Li 2015), multiple methods to train a dialog manager via supervised learning were proposed. Wen et al. (2016) propose a neural-network-based trainable dialog system along with a new way of collecting dialog data. Bordes and Weston (2016) report an end-to-end dialog system based on Memory Networks that can achieve promising, yet imperfect, performance. Eric and Manning (2017) describe a copy-augmented sequence-to-sequence architecture that can also achieve good performance for dialog management.

Dialog managers can also be learned via reinforcement learning (Levin, Pieraccini, and Eckert 1998; Williams and Young 2007; Young et al. 2013). Specifically, the learning processes will be modeled as POMDPs. The dialog manager will directly interact with users and then, in a process of "trial and error", it can learn from delayed rewards. However, learning a dialog manager from scratch via reinforcement learning may be risky and infeasible for industrial bots. Such processes may conduct random explorations, and even sometimes fail to learn an acceptable result, as shown in experiments by Williams and Zweig (2016).

Recently researchers also proposed to learn a dialog manager by combining both supervised learning and reinforcement learning. Zhao and Eskenazi (2016) propose an algorithm that combines the strengths of both to achieve faster learning speed. Williams and Zweig (2016) report that pretraining with supervised learning can substantially accelerate the learning rate of reinforcement learning. Su et al. (2016) demonstrate that supervised learning is effective, and using reinforcement learning after supervised learning can further achieve performance improvements, especially in high-noise conditions.

In this paper, we propose a new dialog manager learning framework termed CoChat. CoChat models the collaboration between bots and human workers for task completion, which, to the best of our knowledge, has not been considered before. CoChat not only combines supervised learning and reinforcement learning as previous works, but also combines online learning to better learn from human workers during bot-worker collaboration. Moreover, unlike previous works that assume the dialog manager to have a fixed action set, CoChat supports new actions which are introduced by human workers. Since new actions generally have just few occurrences, learning how to select them is essentially a one-shot learning problem (Santoro et al. 2016; Graves, Wayne, and Danihelka 2014). To handle that, we propose a memory-enhanced hierarchical RNN model to implement CoChat, i.e., MemHRNN. Neural networks with an external memory have been successfully applied to various NLP problems like machine translation (Tang et al. 2016; Wang et al. 2016) to handle similar challenges.

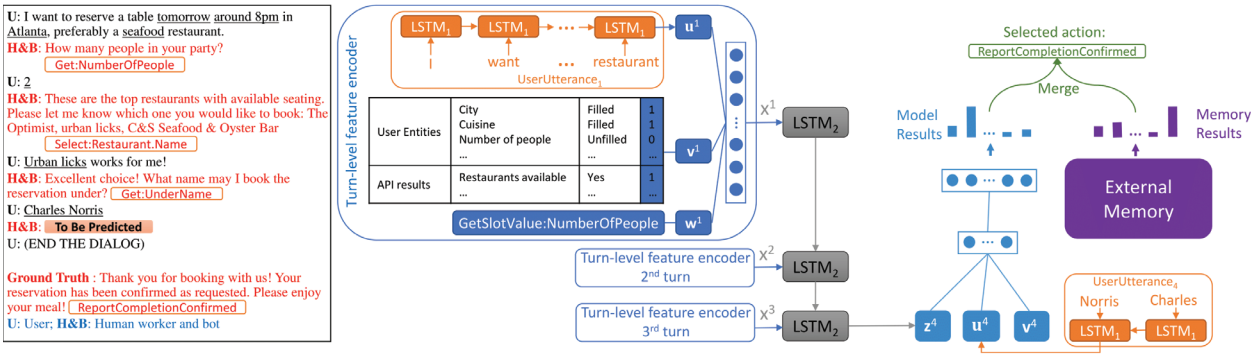


Figure 2: Illustration of the proposed MemHRNN model for implementing the CoChat framework and learning a dialog manager, along with a real complete example from a restaurant booking task. Here underlined words are labeled entities, and framed words are action labels of the corresponding utterances.

CoChat Framework

As illustrated in Fig. 1, our proposed learning framework, CoChat, enables the collaboration between bot and human workers to minimize the risk of failures and ensure user satisfaction. Meanwhile, it helps to minimize human workers' workload during collaboration. In brief, CoChat combines supervised learning from available labelled logs, online learning from feedback given by human workers, and reinforcement learning from delayed rewards/feedback given by users, so as to continuously improve the dialog manager.

Specifically, with a small amount of available labeled user-worker dialogs, the dialog manager is firstly initialized via supervised learning to achieve a higher starting point.

Then the dialog manager is put online to suggest actions to human workers, enabling collaboration and helping them work more efficiently. Generally, human workers can effortlessly choose a suggested action, or input a new action if the suggested ones are inappropriate. These feedback, including the acceptance/rejection of a suggested action and the newly input actions, are utilized to further enhance the dialog manager via online learning. Particularly, when human workers input new actions, CoChat is also expected to learn them for better suggesting actions and handling future similar cases. Actually, introducing new actions can raise one-shot learning challenges. Our proposed model to implement CoChat, i.e., MemHRNN, leverages an external memory to handle such challenges, as elaborated below.

When human workers are unavailable (e.g., off work) but users are willing to try the bot, the dialog manager can directly interact with users. In those cases, the dialog manager is further improved via reinforcement learning from the delayed rewards/feedback given by users, which are related to the success/failure of task completion, user satisfaction, etc.

In CoChat, after initialization via supervised learning, online learning and reinforcement learning are conducted by turns, depending on whether human workers are available to collaborate with the bot or not. Each step can gain further improvements based on the previous ones. Hence, the dialog manager is continuously improved via supervised learning, online learning and reinforcement learning.

$\mathbf{u}^m$	Feature vector of the user utterance, encoded by $LSTM_1$
$\mathbf{v}^m$	Feature vector of the entity form
$\mathbf{w}^m$	1-hot vector of the taken action
$\mathbf{x}^m$	Integrated feature vector of the turn, encoded by *turn-level feature encoder*
$\mathbf{z}^m$	Feature vector of the previous $(m-1)$ turns, encoded by $LSTM_2$

Table 1: Feature vectors related to the mth dialog turn

Proposed MemHRNN to Implement CoChat

To implement the CoChat framework, we propose a memory-enhanced hierarchical RNN model termed MemHRNN. MemHRNN consists of a hierarchical RNN based learning model (i.e., HRNN) and an external memory. The external memory only works during online learning, while HRNN works through all learning processes. Denoting all parameters in HRNN as θ, θ is shared and continuously optimized among all learning processes. We will elaborate on HRNN and all learning processes in the following subsections.

HRNN for All Learning Processes

The HRNN in MemHRNN consists of *turn-level feature encoder* and *action selector*. For any dialog, it can be split into several turns, each consisting of a user utterance and the action taken by human workers or the bot. And here the *turn-level feature encoder* will be applied to each turn in a dialog to get its corresponding feature vector. Then the *action selector* takes such turn-level feature vectors as input to decide which action to choose. Below we will introduce both, with the important notations summarized in Table 1.

Turn-level Feature Encoder Suppose at the mth turn, the user utterance u^m is defined as a word sequence $u^m = \{w^{m,1}, w^{m,2}, \ldots, w^{m,n_m}\}$, with n_m being the quantity of words. Then an Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997), denoted as $LSTM_1$, is utilized as follows to encode the embedding vector of each word one by one.

$$\mathbf{u}^{m,j} = LSTM_1(\omega^{m,j}, \mathbf{u}^{m,j-1}) \quad (1)$$

where $\omega^{m,j}$ is the embedding vector of the word $w^{m,j}$ and $\mathbf{u}^{m,j-1}$ is the encoded feature vector for the previous $(j-1)$

words. And thus the feature vector $\mathbf{u}^m$ for u^m will be the last output of $LSTM_1$, i.e., $\mathbf{u}^m = \mathbf{u}^{m,n_m}$.

For the mth turn, we also consider the following features. 1) *Entity form* collects information for completing the task. As shown in Fig. 2, it generally contains two kinds of information, i.e., entities extracted from user utterances and information returned by API calls. Here we simply encode the entity form as a binary (i.e., 0 or 1) feature vector $\mathbf{v}^m$, indicating whether an entity is extracted or whether an API call returns true / available data. 2) *Taken action* at the mth turn is represented as a n_a dimensional 1-hot vector $\mathbf{w}^m$, with n_a being the number of actions. Only the entry corresponding to the taken action is set as 1, and others are 0.

For the mth turn, $\mathbf{u}^m$, $\mathbf{v}^m$ and $\mathbf{w}^m$ are concatenated as a feature vector $\hat{\mathbf{x}}^m = [\mathbf{u}^m, \mathbf{v}^m, \mathbf{w}^m]$, and then fed to a fully-connected layer FN to learn a new feature vector $\mathbf{x}^m$:

$$\mathbf{x}^m = FN(\hat{\mathbf{x}}^m). \tag{2}$$

The reason for using $\mathbf{x}^m$ instead of $\hat{\mathbf{x}}^m$ is that using FN can help to better reserve the accumulated knowledge, especially when new actions are involved and HRNN needs to be changed. This will be explained with more details later in the subsection about online learning.

Action Selector The action selector takes the historical dialog turns and the current one as input, and then predicts the probability of selecting each candidate action.

Suppose the current dialog turn is the mth one. Like hierarchical LSTM (Li, Luong, and Dan 2015; Xie et al. 2016; Yang; and Huang 2016; Sordoni et al. 2015), another LSTM, denoted as $LSTM_2$, is utilized as follows to encode the previous $(m-1)$ dialog turns one by one into a vector $\mathbf{z}^m$.

$$\mathbf{z}^m = LSTM_2(\mathbf{x}^{m-1}, \mathbf{z}^{m-1}) \tag{3}$$

where $\mathbf{z}^{m-1}$ is the encoded feature vector of the previous $(m-2)$ turns and can be derived in an identical way recursively. For the current turn, since no action has been taken, we can only derive the feature vector $\mathbf{u}^m$ of the user utterance and the feature vector $\mathbf{v}^m$ of the entity form, as defined in Table 1. Then $\mathbf{z}^m$, $\mathbf{u}^m$ and $\mathbf{v}^m$ are concatenated as an integrated vector $\mathbf{s}^m = [\mathbf{z}^m, \mathbf{u}^m, \mathbf{v}^m]$ and fed to a two-layer fully-connected neural network FN_2. Note that we utilize a *softmax* function for the 2nd layer to derive the probability distribution $\mathbf{p}^m$ for selecting all candidate actions:

$$\mathbf{p}^m = FN_2(\mathbf{s}^m) \tag{4}$$

Learning from Logs: Supervised Learning

Given available labelled user-worker dialog logs, the dialog manager is initialized via supervised learning to imitate human workers. Specifically, in each dialog turn of the logs, the chosen action by human workers is taken as the ground-truth one. And it is denoted by a one-hot vector $\mathbf{y}^m \in \{0,1\}^{n_a}$, where m is the dialog turn index and n_a is the number of candidate actions. Using HRNN for supervised learning, we aim to minimize the cross-entropy between $\mathbf{y}^m$ and the predicted probability distribution $\mathbf{p}^m$ over all actions in formula (4), as the loss function below shows.

$$\mathcal{L}_{SL}(\theta) = \frac{1}{n_d}\sum_{m=1}^{n_d}\mathcal{E}(\mathbf{p}^m) \quad s.t., \mathcal{E}(\mathbf{p}^m) = -\sum_{i=1}^{n_a}\mathbf{y}_i^m \log \mathbf{p}_i^m \tag{5}$$

where θ denotes all the parameters of HRNN, and n_d is the number of dialog turns in the labelled logs.

Learning from Human Workers: Online Learning

When the bot starts working with human workers, its dialog manager can be improved using feedback from human workers. Specifically, at each conversation turn, the bot suggests k actions to the human worker. These k actions are decided by the HRNN together with the external memory. Human workers can either accept one (i.e., select one of the top n suggested actions and directly applied to interact with the user) or refuse all to input another action that can better handle the dialog state. Then the HRNN and the external memory are updated accordingly.

Updating HRNN The HRNN is updated differently according to whether the human workers input a new action outside the existing action set or not.

Case 1: When the human workers accept a suggested action or input an existing candidate action, such a feedback can be treated as a new labelled log and directly utilized to update the dialog manager. The model is continuously updated with such newly collected logs, and it *makes the dialog manager better fit the time-varying action choices*. Given a small batch of newly collected logs, we aim to minimize the following loss function for online learning.

$$\mathcal{L}_{OL}(\theta) = \frac{1}{n_o}\sum_{m=1}^{n_o}\mathcal{E}(\mathbf{p}^m) + \lambda||\theta - \theta'||_2 \tag{6}$$

where n_o is the number of dialog turns in the small batch of newly collected logs, $\mathcal{E}(\mathbf{p}^m)$ is the cross-entropy defined in the same way as that in formula (5), θ denotes all parameters of HRNN, and θ' denotes the values of θ before collecting the small batch of logs. Considering that the dialog management policy changes gradually in most cases, we add the 2nd term to avoid sharp changes of θ during online learning which can also make the learning process more stable. Here λ is a balancing factor, and we empirically set it as 0.05.

Case 2: If human workers input a new action, it will be added to the set of candidate actions, and thus the network architecture of HRNN needs to be updated accordingly. Specifically, the input layer of the turn-level feature encoder and the output layer of HRNN will be changed.

For the input layer of the turn-level feature encoder, the dimensionality of the feature vector $\mathbf{w}^m$ in Table 1 will be extended by one for the new action, while the output dimensionality of FN is unchanged. Then we add full connections between the newly added dimension and the output units of FN. Original weights of FN are all reserved, while the weights on the newly added connections are initialized as random values close to 0, so that FN can still work nearly the same as before when a new action is just added. As the output dimensionality of FN is unchanged, the architecture of $LSTM_2$, which is the key to encoding historical dialog turns, need not be changed. Hence the influence of adding new actions on HRNN is substantially reduced, and thus the learning model can reserve as much accumulated knowledge as possible. That is also why FN is introduced.

As for the output layer of HRNN, i.e., the last layer of FN_2, we add a new dimension corresponding to the new action, and add full connections between it and the input units of the layer. Similarly, to make HRNN work nearly the same as before when a new action is just added, we also initialize the weights on the newly added connections as random values close to 0.

After changing the architecture, HRNN will be updated in the same way as Case 1.

Leveraging External Memory The external memory is introduced for handling the one-shot learning challenges caused by instantly introducing new actions. Specifically, the occurrences of a new action are recorded in an external memory M, and then used together with the HRNN to suggest actions. The usage of the external memory are detailed below, with the dialog turn index m omitted for simplicity.

Actions outside the pre-defined action set in the supervised learning period are regarded as new actions. And the occurrence of a new action a is recorded as a key-value pair $\langle a, \mathbf{r} \rangle$ in the memory, where $\mathbf{r}$ denotes the dialog state for the occurrence. Since the representation derived by HRNN for a dialog state will always vary as its parameters get updated during optimization, here we utilize a simplified way to derive $\mathbf{r}$ for the memory. Specifically, $\mathbf{r}$ consists of two parts, i.e., $\mathbf{r} = [\mathbf{v}, \mathbf{c}]$ where $\mathbf{v}$ is the entity form in Table 1 and $\mathbf{c}$ is a context vector derived as the mean pooling of the embedding vectors of words in the last user utterance.

Moreover, when the occurrence of a new action is above a predefined threshold, its corresponding records will be erased from the memory. Actually, for those erased new actions, their occurrences are supposed to be sufficient for the HRNN to learn valid strategies regarding them, making the one-shot learning challenges well relieved, and thus the help from the external memory is negligible for them. Here we denote the actions recorded in the external memory as A.

Given a dialog turn, HRNN can directly predict a probability distribution $\mathbf{p}$ as formula (4) for selecting each candidate action. Meanwhile, for the memory, suppose the state of the dialog turn is denoted as $\hat{\mathbf{r}} = [\hat{\mathbf{v}}, \hat{\mathbf{c}}]$ and derived in the same way as those recorded. We can leverage the external memory together with HRNN as follows.

STEP 1: We calculate the similarity between $\hat{\mathbf{r}}$ and any recorded occurrence $\mathbf{r}$ as follows.

$$Sim(\hat{\mathbf{r}}, \mathbf{r}) = \exp\left(-\frac{\alpha\|\hat{\mathbf{v}} - \mathbf{v}\|_2 + (1-\alpha)\|\hat{\mathbf{c}} - \mathbf{c}\|_2}{2\sigma^2}\right) \quad (7)$$

where α is a balancing factor empirically set as 0.1 in our experiments, and σ is a smoothing parameter.

STEP 2: We define a similarity vector $\mathbf{h}$ over *all candidate actions*, with $\mathbf{h}_i$ defined as follows.

$$\mathbf{h}_i = \begin{cases} 0 & \text{if } a_i \notin A, \\ \max_{\mathbf{r} \in R(a_i)} Sim(\hat{\mathbf{r}}, \mathbf{r}) & \text{if } a_i \in A. \end{cases} \quad (8)$$

where a_i is the ith candidate action, and $R(a_i)$ denotes all the recorded occurrences of a_i.

As only few occurrences of a new action are recorded in the memory, they are supposed to be used in a restrict manner with less generalization. Namely, we may probably select a new action in A only when the dialog state $\hat{\mathbf{r}}$ is very similar to one of its corresponding recorded occurrences. Otherwise, we may prefer other actions. To realize that, we derive a discounted factor $\beta = sigmoid(\max(\mathbf{h}))$, where $max(\mathbf{h})$ is the maximum value of $\mathbf{h}$, and $sigmoid$ denotes a sigmoid function. Then we derive a *discounted* probability distribution $\mathbf{q}$ as follows for selecting new actions in A.

$$\mathbf{q}_i = \beta \frac{\exp(\mathbf{h}_i)}{\sum_j \exp(\mathbf{h}_j)} \quad (9)$$

STEP 3: We merge $\mathbf{q}$ with the probability distribution $\mathbf{p}$ from HRNN into an integrated one $\hat{\mathbf{p}}$ as follows.

$$\hat{\mathbf{p}}_i = \begin{cases} \mathbf{p}_i & \text{if } a_i \notin A, \\ \mathbf{q}_i & \text{if } a_i \in A. \end{cases} \quad (10)$$

Then MemHRNN suggests those actions with the top k highest probabilities in $\hat{\mathbf{p}}$ to human workers.

Learning from Users: Reinforcement Learning

When the bot have chances to directly interact with users and choose actions by itself, its dialog manager can be further improved via reinforcement learning with the delayed rewards / feedback from users, which are generally received after the whole dialog finishes and are mostly related to the success / failure of task completion, user satisfaction, etc.

In reinforcement learning, at any dialog state s^m with m denoting the corresponding dialog turn index, the dialog manager aims to select an action a^m that maximizes the cumulative future rewards. Hence reinforcement learning defines an optimal action-value function as below.

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[r^{(m)} + \gamma r^{(m+1)} + \cdots | s^m = s, a^m = a, \pi]$$

where $\gamma \in [0, 1]$ is a discounted factor, $r^{(m)}$ and $r^{(m+j)} (j = 1, 2, \ldots)$ are respectively the received rewards at the mth dialog turn and the subsequent turns, π is the optimal dialog management policy, and thus $Q^*(s, a)$ is the maximal discounted cumulative rewards of taking the action a as a^m when s^m is observed to be the dialog state s.

Like deep Q-network (DQN) (Mnih et al. 2015), we utilize the HRNN, which has already been pre-trained via supervised learning and online learning, to approximate the optimal action-value function, denoted as $Q(s, a; \theta)$ with θ being all parameters of HRNN. Then we minimize the following loss function to perform Q-learning (Mnih et al. 2015) for updating θ iteratively.

$$\mathcal{L}_{RL}(\theta) = \mathbb{E}_{s,a,r,s'}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]$$

where θ^- denotes parameters of the target network in the algorithm (Mnih et al. 2015), s' and r are respectively the new dialog state and the received reward after taking the action a at the state s, and a' is a new action for s'.

Summary

Though supervised learning, online learning and reinforcement learning can have different loss functions (i.e., $\mathcal{L}_{SL}$, $\mathcal{L}_{OL}$ and $\mathcal{L}_{RL}$) and update the model parameters in different ways, they use an identical neural network (i.e., θ) to share their learned knowledge and make continuous improvements for the dialog manager based on each other.

	Restaurant	Movie
# average dialog turns	3.8	3.5
# max dialog turns	8	10
# actions	49	70
# dialogs	1490	1490

Table 2: Basic statistics of collected datasets.

Experiments

Dataset and User Simulator

We hired human workers and users who are familiar with two realistic tasks, i.e., booking restaurants and booking movie tickets, and collected their dialogs to build two datasets for our experiments. A complete example dialog is shown in Fig. 2, and some basic statistics of both datasets are reported in Table 2. Such realistic datasets can better evaluate the practicability of learning frameworks and models for dialog manager. Generally, we define an action as one or more operations on some (possibly empty) slots. Operations include "Get", "Select", "ChitChat", etc., as illustrated in Fig. 2. Slot values can be obtained from API calls, and different values (e.g., different restaurant names) can correspond to one action. The full list of actions is available in the supplementary material[1]. When collecting both datasets, we also find that the dialog management policies usually change with time and the action sets will grow larger with time. In our experiments, all dialogs are sequentially fed to the learning processes based on their collecting timestamps.

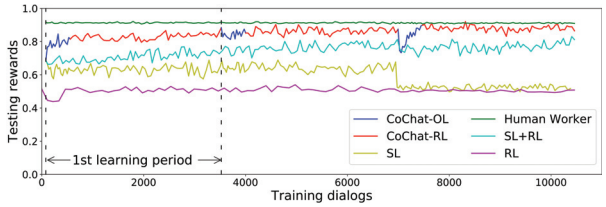
As numerous interactions with users are required for experiments, we follow previous works (Schatzmann et al. 2007) and build user simulators with collected dialogs. Then we use them to generate *more* dialogs for training dialog manager via reinforcement learning, and testing each learned dialog manager. Note that to better imitate real users, the user simulators would be rebuilt with time. Similar with (Lipton et al. 2016), we set the rewards of reinforcement learning as follows: a fixed punishment (i.e., -0.025) for every turn, a large reward (i.e., 1) for successful task completions and a small one (i.e., 0.5) for failures.

Evaluation for the *CoChat* Framework

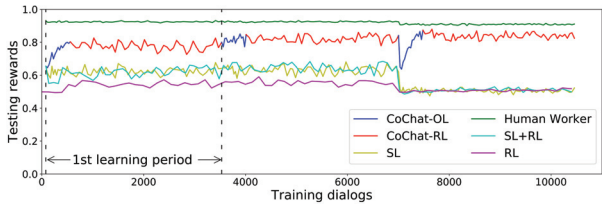
We compare different frameworks by training the proposed MemHRNN under them, to see if *CoChat* can better maximize user satisfaction and minimize workers' workload.

Settings To clarify each learning process in *CoChat*, we conduct experiments as follows. Firstly, to track real-world situations, the dialog manager is trained via supervised learning with a small amount of collected dialogs (i.e., the first 50). Then online learning and reinforcement learning are conducted alternately on the subsequent dialogs. An online learning process and its subsequent reinforcement learning process compose a *learning period*. In each learning period, online learning is conducted on the collected dialogs, while reinforcement learning is conducted using a user simulator built with the dialogs in online learning.

[1]Available at: http://irip.buaa.edu.cn/Research/luoxufang/CoChat_supp.pdf



(a) Restaurant.



(b) Movie.

Figure 3: Curves tracking the testing rewards of human workers and learned dialog managers.

Dataset	Human Worker	*CoChat*	*SL*	*RL*	*SL+RL*
Restaurant	0.911	**0.884**	0.523	0.501	0.791
Movie	0.908	**0.841**	0.505	0.512	0.511

Table 3: Testing rewards of human workers and learned dialog managers at the ending stage.

Dataset	Frameworks	1^{st}	2^{nd}	3^{rd}
Restaurant	*CoChat*	**89.91%**	**89.87%**	**86.87%**
	SL	82.07%	80.27%	80.10%
Movie	*CoChat*	**82.49%**	**83.82%**	**78.73%**
	SL	66.81%	65.60%	51.14%

Table 4: Top 5 hit rates for action suggestion in 3 online learning processes.

Compared frameworks include reinforcement learning (*RL*), supervised learning (*SL*), and combination of both (*SL+RL*). In *RL*, dialog manager is trained using only reinforcement learning from scratch. In *SL*, dialog manager is trained with the first 50 collected dialogs via supervised learning and won't be updated. *SL+RL* uses supervised learning for initialization and then applies reinforcement learning. In each learning period, dialog managers of the *RL* framework and the *SL+RL* framework are trained using the same user simulator as that used in *CoChat*.

As for the performance metrics, user satisfaction is measured with *reward*. Specifically, we define the testing reward as the average reward per dialog gained by a framework after it interacts with the user simulator for 100 dialogs. For measuring workers' workload, we focus on cases where the bot collaborates with human workers and suggests actions for them. We use *top 5 hit rate*, i.e., the percentage of dialog turns where the workers accept an action in the top 5 suggested ones, to measure the action suggestion performance. Higher hit rates mean lighter workload.

Results All the curves tracking testing rewards of different frameworks are shown in Fig. 3. We can observe that: 1) *human workers* outperform all learned dialog managers in

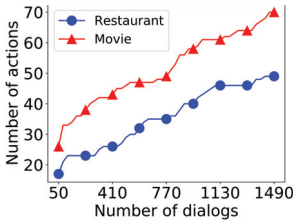


Figure 4: Curves tracking how the number of actions grows as more and more dialogs are collected.

	# dialogs with new actions	Ratio
Restaurant	305	21.18%
Movie	278	19.31%

Figure 5: The number of dialogs with new actions and the corresponding percentages in both datasets.

maximizing user satisfaction, and thus the proposed *CoChat* involving human workers in the loop is reasonable and able to maximize user satisfaction when workers are available; 2) Among all learned dialog managers, when they interact with users directly, the one learned via *CoChat* outperforms those via other frameworks, well demonstrating the superiority of *CoChat*; 3) Performance of the learned dialog manager via *CoChat* may drop during online learning (i.e., online learning in *CoChat*) due to changes of dialog management policies or new actions, but it can fast recover from the drop, rather than keep declined as *SL*, meaning that *CoChat* can better fit to new policies/actions; 4) Putting aside policy changes or new actions, generally the supervised learning, online learning and reinforcement learning in *CoChat* together make the dialog manager continuously improved.

Average testing rewards of different dialog managers at the ending stage (i.e., the last 400 dialogs) are reported in Table 3. On both tasks, after 3 learning periods, the rewards gained by the dialog manager learned via *CoChat* are respectively 97.04% and 92.62% of those by human workers.

We report the top 5 hit rates for action suggestion in the 3 online learning processes of *CoChat*, compared with those of *SL* in Table 4. It can be seen that, $80\%\sim90\%$ of workers' workload can be relieved by the bot learned via CoChat when it suggests 5 actions.

Analyses about New Actions

Statistics about New Actions To clarify the necessity of handling new action challenges, we further analyze some statistics about new actions in the real-world datasets.

Fig. 4 presents how the number of actions grows as more and more dialogs are collected for the two datasets. It can be seen that new actions keep coming up with time.

Besides, Table 5 shows that the percentages of dialogs with new actions are about 20% in both datasets, meaning that new actions are also frequently used in the subsequent dialogs after their coming up. Therefore, well handling new actions is necessary for learning a good dialog manager.

Models	Restaurant	Movie
MemHRNN	**91.35%**	**86.32%**
HCN (Williams, Asadi, and Zweig 2017)	89.47%	82.15%
HLSTM (Xie et al. 2016)	87.80%	81.36%
Liu and Lane (2017)	87.97%	81.66%

Table 5: Top 5 hit rates of compared models for action suggestion on both datasets.

Models	Restaurant	Movie
MemHRNN	**91.35%**	**86.32%**
HRNN (NO Memory)	90.34%	84.82%

Table 6: Top 5 hit rates of MemHRNN and HRNN (no memory) for action suggestion during online learning.

MemHRNN for Handling New Actions To validate whether the proposed MemHRNN can better handle new actions in online learning processes, we compare it with other state-of-the-art models in the same learning processes.

Settings Here all models are firstly trained with the first 50 dialogs, and then online learning is conducted with the subsequent 1440 collected dialogs. New actions will be involved in this learning process. And for compared models that cannot handle new actions, the dialog turns with new actions will be ignored for training.

Results The top 5 hit rates of compared models for action suggestion are reported in Table 5. It can be seen that, by allowing introducing new actions and well handling the corresponding one-shot learning challenges, the proposed MemHRNN can better suggest actions for human workers to reduce their workload.

Superiority of Leveraging External Memory

Settings Here the experimental settings are the same as the experiment above. And the proposed MemHRNN is compared with the HRNN model using no memory, so as to see whether the external memory can really help to tackle the one-shot learning challenges caused by new actions.

Results The top 5 hit rates of the proposed MemHRNN and the HRNN using no memory for action suggestion are reported in Table 6, which demonstrate that the introduced external memory really enhances MemHRNN to handle the one-shot learning challenges of new actions. For detailed case analyses, one can refer to the supplemental material.

Conclusions

In this paper, we propose a dialog manager learning framework CoChat that models the collaboration between bots and human workers for task completion. CoChat aims to maximize user satisfaction and minimize the workers' workload. Moreover, CoChat combines supervised learning, online learning and reinforcement learning to continuously improve the to-be-learned dialog manager. Here we also propose a memory-enhanced hierarchical RNN model termed MemHRNN to implement the CoChat framework. Particularly, MemHRNN can tackle the one-shot learning challenges caused by instantly introducing new actions, with the

help of an external memory. Extensive experiments on real-world datasets also well demonstrate the effectiveness of the proposed CoChat framework and the MemHRNN model.

Acknowledgments

This work is partly supported by the National Natural Science Foundation of China (No. 61421003) and the Microsoft Research Asia Collaborative Program (FY17-RES-THEME-033).

References

Bordes, A., and Weston, J. 2016. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*.

Eric, M., and Manning, C. D. 2017. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. *arXiv preprint arXiv:1701.04024*.

Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

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

Lee, C.-J.; Jung, S.-K.; Kim, K.-D.; Lee, D.-H.; and Lee, G. G.-B. 2010. Recent approaches to dialog management for spoken dialog systems. *Journal of Computing Science and Engineering* 4(1):1–22.

Levin, E.; Pieraccini, R.; and Eckert, W. 1998. Using markov decision process for learning dialogue strategies. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 1, 201–204. IEEE.

Li, J.; Luong, M. T.; and Dan, J. 2015. A hierarchical neural autoencoder for paragraphs and documents. *Computer Science*.

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, B., and Lane, I. 2017. An end-to-end trainable neural network model with belief tracking for task-oriented dialog. *Proc. Interspeech 2017* 2506–2510.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; and Ostrovski, G. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–33.

Santoro, A.; Bartunov, S.; Botvinick, M.; Wierstra, D.; and Lillicrap, T. 2016. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, 1842–1850.

Saunders, W.; Sastry, G.; Stuhlmueller, A.; and Evans, O. 2017. Trial without error: Towards safe reinforcement learning via human intervention. *arXiv preprint arXiv:1707.05173*.

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.

Shang, L.; Lu, Z.; and Li, H. 2015. Neural responding machine for short-text conversation. *arXiv preprint arXiv:1503.02364*.

Sordoni, A.; Bengio, Y.; Vahabi, H.; Lioma, C.; Grue Simonsen, J.; and Nie, J.-Y. 2015. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 553–562. ACM.

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, P.-H.; Budzianowski, P.; Ultes, S.; Gasic, M.; and Young, S. 2017. Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management. *arXiv preprint arXiv:1707.00130*.

Tang, Y.; Meng, F.; Lu, Z.; Li, H.; and Yu, P. L. 2016. Neural machine translation with external phrase memory. *arXiv preprint arXiv:1606.01792*.

Vinyals, O., and Le, Q. 2015. A neural conversational model. *Computer Science*.

Wang, M.; Lu, Z.; Li, H.; and Liu, Q. 2016. Memory-enhanced decoder for neural machine translation. *arXiv preprint arXiv:1606.02003*.

Wen, T.-H.; Gasic, M.; Mrksic, N.; Rojas-Barahona, L. M.; Su, P.-H.; Ultes, S.; Vandyke, D.; and Young, S. 2016. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*.

Williams, J. D., and Young, S. 2007. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language* 21(2):393–422.

Williams, J. D., and Zweig, G. 2016. End-to-end lstm-based dialog control optimized with supervised and reinforcement learning. *arXiv preprint arXiv:1606.01269*.

Williams, J. D.; Asadi, K.; and Zweig, G. 2017. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. *arXiv preprint arXiv:1702.03274*.

Xie, R.; Liu, Z.; Yan, R.; and Sun, M. 2016. Neural emoji recommendation in dialogue systems. *arXiv preprint arXiv:1612.04609*.

Yang;, M.-H. S. K.-Y. H. T.-H., and Huang, T.-C. 2016. Dialog state tracking for interview coaching using two-level lstm.

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

Young, S. J. 2006. Using pomdps for dialog management. In *SLT*, 8–13.

Zhao, T., and Eskenazi, M. 2016. Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. In *17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 1.