End-to-End Reinforcement Learning of Dialogue Agents for Information Access

Bhuwan Dhingra[⋆∗] **Lihong Li**[†] **Xiujun Li**[†] **Jianfeng Gao**[†]
Yun-Nung Chen[‡∗] **Faisal Ahmed**[†] **Li Deng**[†]
[⋆]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[†]Microsoft Research, Redmond, WA, USA
[‡]National Taiwan University, Taipei, Taiwan
[⋆]bdhingra@andrew.cmu.edu [†]{lihongli,xiul,jfgao,vivic,fiahmed,deng}@microsoft.com

arXiv:1609.00777v2 [cs.CL] 31 Oct 2016

Abstract

This paper proposes *KB-InfoBot*—a dialogue agent that provides users with an entity from a knowledge base (KB) by interactively asking for its attributes. All components of the KB-InfoBot are trained in an end-to-end fashion using reinforcement learning. Goal-oriented dialogue systems typically need to interact with an external database to access real-world knowledge (e.g., movies playing in a city). Previous systems achieved this by issuing a symbolic query to the database and adding retrieved results to the dialogue state. However, such symbolic operations break the differentiability of the system and prevent end-to-end training of neural dialogue agents. In this paper, we address this limitation by replacing symbolic queries with an induced "soft" posterior distribution over the KB that indicates which entities the user is interested in. We also provide a modified version of the episodic REINFORCE algorithm, which allows the KB-InfoBot to explore and learn both the policy for selecting dialogue acts and the posterior over the KB for retrieving the correct entities. Experimental results show that the end-to-end trained KB-InfoBot outperforms competitive rule-based baselines, as well as agents which are not end-to-end trainable.

1 Introduction

Goal-oriented dialogue systems help users complete specific tasks, such as booking a flight or searching a database, by interacting with them via natural language. In this work, we present *KB-InfoBot*, a dialogue agent that identifies entities of interest to the user from a knowledge base (KB), by interactively asking for attributes of that entity which helps constrain the search. Such an agent finds application in interactive search settings. Figure 1 shows a dialogue example between a user searching for a movie and the proposed KB-InfoBot.

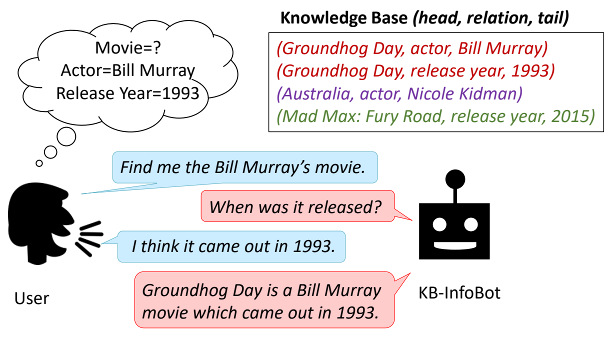

Knowledge Base (head, relation, tail)
(Groundhog Day, actor, Bill Murray)
(Groundhog Day, release year, 1993)
(Australia, actor, Nicole Kidman)
(Mad Max: Fury Road, release year, 2015)
Movie=?
Actor=Bill Murray
Release Year=1993
Find me the Bill Murray's movie.
When was it released?
I think it came out in 1993.
Groundhog Day is a Bill Murray movie which came out in 1993.
User
KB-InfoBot

Figure 1: A dialogue example between a user looking for a movie and the KB-InfoBot. The knowledge base is shown above the KB-InfoBot.

A typical goal-oriented dialogue system consists of four basic components: a *language understanding* (LU) module for identifying user intents and extracting associated slots (Yao et al., 2014; Hakkani-Tür et al., 2016; Chen et al., 2016), a dialogue *state tracker* which tracks the user goal and dialogue history (Henderson et al., 2014; Henderson, 2015), a *dialogue policy* which selects the next system action based on the current state (Young et al., 2013), and a *natural language generator* (NLG) for converting dialogue acts into natural language (Wen et al., 2015; Wen et al., 2016a). For successful completion of user goals, it is also necessary to equip

[∗]Work completed while BD and YNC were with Microsoft.

the dialogue policy with real-world knowledge from a database. Previous end-to-end systems achieved this by constructing a symbolic query from the current belief states of the agent and retrieving results from the database which match the query (Wen et al., 2016b; Williams and Zweig, 2016; Zhao and Eskenazi, 2016). Unfortunately, such operations make the model non-differentiable, and various components in a dialogue system are usually trained separately.

In our work, we replace SQL-like queries with a probabilistic framework for inducing a posterior distribution of the user target over KB entities. We build this distribution from the belief tracker multinomials over attribute-values and binomial probabilities of the user not knowing the value of an attribute. The policy network receives as input this full distribution to select its next action. In addition to making the model end-to-end trainable, this operation also provides a principled framework to propagate the uncertainty inherent in language understanding to the dialogue policy making the agent robust to LU errors.

Our entire model is differentiable, which means that in theory our system can be trained completely end-to-end using only a reinforcement signal from the user that indicates whether a dialogue is successful or not. However, in practice, we find that with random initialization the agent is unable to see any rewards if the database is large; even when it does, *credit assignment* is tough. Hence, at the beginning of training, we first have an imitation-learning phase (Argall et al., 2009) where both the belief tracker and policy network are trained to mimic a rule-based agent. Then, on switching to reinforcement learning, the agent is able to improve further and increase its average reward. Such a bootstrapping approach has been shown effective when applying reinforcement learning to solve hard problems, especially those with long decision horizons (Silver et al., 2016).

Our key contributions are three-fold. First, we present a probabilistic framework for inducing a posterior distribution over the entities in a knowledge base. Second, we use the above framework to develop, to our knowledge, the first fully end-to-end differentiable model of a multi-turn information providing dialogue agent (KB-InfoBot), whose parameters can be tuned using standard gradient descent methods. Third, we present a modified version of the episodic REINFORCE (Williams, 1992) update rule for training the above model based on user feedback, which allows the agent to explore both the set of possible dialogue acts at each turn and the set of possible entity results from the KB at the final turn.

2 Related Work

Statistical goal-oriented dialogue systems have long been modeled as partially observable Markov decision processes (POMDPs) (Young et al., 2013), and are trained using reinforcement learning based on user feedback. Recently, there has been growing interest in designing "end-to-end" systems, which combine feature extraction and policy optimization using deep neural networks, with the aim of eliminating the need of hand-crafted representations. We discuss these works below and highlight their methods of interfacing with the external database.

Cuayáhuitl (2016) proposed *SimpleDS*, which uses a multi-layer feed-forward network to directly map environment states to agent actions. The network is trained using Q-learning and a simulated user; however it does not interact with a structured database, leaving that task to a server, which may be suboptimal as we show in our experiments below.

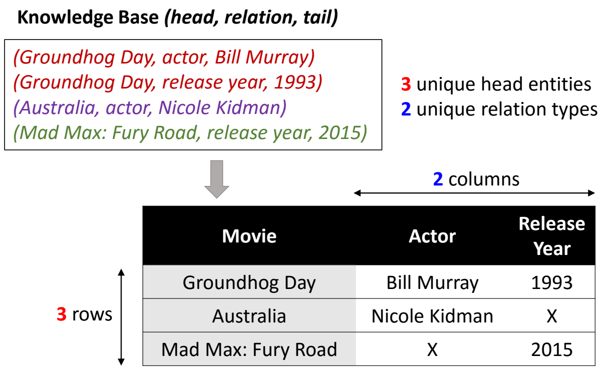
Wen et al. (2016b) introduced a modular dialogue agent, which consists of several neural-network-based components trained using supervised learning. One key component is the database operator, which forms a query $q_t = \cup_{s' \in S_I} \arg\max_v p_{s'}^t$, where $p_{s'}^t$ are distributions over the possible values of each slot and are output from the belief tracker. The query is issued to the database which returns a list of the matched entries. We refer to this operation henceforth as a *Hard-KB* lookup. Hard-KB lookup breaks the differentiability of the whole system, and as a result training of various components of the dialogue system needs to be performed separately. The intent network and belief trackers are trained using supervised labels specifically collected for them; while the policy network and generation network are trained separately on the system utterances. In this paper, we retain modularity of the network by keeping the belief trackers separate, but replace the query with a *differentiable lookup* over the database which computes a posterior distribution denoting the probability that the user is looking for a particular entry.

An alternative way for the dialogue agent to interface with the database is by augmenting its action space with predefined API calls (Williams and Zweig, 2016; Zhao and Eskenazi, 2016; Bordes and Weston, 2016). The API calls modify a query hypothesis maintained outside the end-to-end system which is used to retrieve results from this KB. These results are then appended to the next system input based on which the agent selects its next action. The resulting model is end-to-end differentiable, albeit with the database falling out of it. This framework does not deal with uncertainty in language understanding since the query hypothesis can only hold one slot-value at a time. Our approach, on the other hand, directly models the uncertainty to come up with a posterior distribution over entities in the knowledge base.

Wu et al. (2015) recently presented an entropy minimization dialogue management (EMDM) strategy for KB-InfoBots. The agent always asks for the value of the slot with maximum entropy over the remaining entries in the database. This approach is optimal in the absence of LU errors, but suffers from error propagation in their presence. This rule-based policy serves as a baseline to compare to our proposed approach.

Our work is motivated by the neural GenQA (Yin et al., 2016a) and neural enquirer (Yin et al., 2016b) models for querying KB and tables via natural language in a fully "neuralized" way. These works handle single-turn dialogues and are trained using supervised learning, while our model is designed for multi-turn dialogues and trained using reinforcement learning. Moreover, instead of defining an attention distribution directly over the KB entities, which could be very large, we instead induce it from the smaller distributions over each *relation* (or *slot* in dialogue terminology) in the KB. A separate line of work—TensorLog (Cohen, 2016)—investigates reasoning over chains of KB facts in a differentiable manner to retrieve new facts. Instead, our focus is on retrieving entities present in the KB given some of the facts they participate in.

Reinforcement Learning Neural Turing Machines (RL-NTM), introduced by Zaremba and Sutskever (2015), also allow neural controllers to interact with discrete external interfaces. The particular form of interface considered in that work is a one-dimensional memory tape along which a read head can move. Our work is in a similar vein, but assumes a different interface—an entity-centric KB. We exploit the structure of such KBs to provide differentiable access to an KB-InfoBot agent for making decisions.


Knowledge Base *(head, relation, tail)*

(Groundhog Day, actor, Bill Murray)
(Groundhog Day, release year, 1993)
(Australia, actor, Nicole Kidman)
(Mad Max: Fury Road, release year, 2015)

Movie	Actor	Release Year
Groundhog Day	Bill Murray	1993
Australia	Nicole Kidman	X
Mad Max: Fury Road	X	2015

Figure 2: An entity-centric knowledge base, where head entities are movies. **Top:** Conventional (h, r, t) format. **Bottom:** Table format. Missing values are denoted by X.

Li et al. (2016) recently applied deep reinforcement leaning successfully to train non-goal oriented chatbot type dialogue agents. They show that reinforcement learning allows the agent to model long-term rewards and generate more diverse and coherent responses as compared to supervised learning. Chatbot systems, however, typically do not need to interface with an external database, which is the primary focus of this paper.

3 Probabilistic Framework for KB Lookup

In this section we describe a probabilistic framework for querying a KB given the agent's beliefs over the slots or attributes in the KB.

3.1 Entity-Centric Knowledge Base (EC-KB)

A Knowledge Base consists of triples of the form (h, r, t), which denotes that *relation* r holds between the *head* h and *tail* t. In this work we assume that the KB-InfoBot has access to a domain-specific entity-centric knowledge base (EC-KB) (Zwicklbauer et al., 2013) where all head entities are of a particular type, and the relations correspond to attributes of these head entities. Examples of the type include movies, persons, or academic papers. Such a KB can be converted to a table format whose rows correspond to the unique head entities, columns correspond to the unique relation types (*slots* henceforth), and some of the entries may be missing. A small example is shown in Figure 2.

3.2 Notations and Assumptions

Let $\mathcal{T}$ denote the KB table described above and $\mathcal{T}_{i,j}$ denote the jth slot-value of ith entity. $1 \leq i \leq N$ and $1 \leq j \leq M$. We let V^j denote the vocabulary of each slot, i.e. the set of all distinct values in j-th column. We denote missing values from the table with a special token and write $\mathcal{T}_{i,j} = \Psi$. $M_j = \{i : \mathcal{T}_{i,j} = \Psi\}$ denotes the set of entities for which the value of slot j is missing. Note that the user may still know the actual value of $\mathcal{T}_{i,j}$, and we assume this lies in V^j. Hence, we do not deal with OOV entities or relations at test time.

The user goal is sampled uniformly $G \sim \mathcal{U}[\{1, ...N\}]$ and points to a particular row in the table $\mathcal{T}$. To make the problem realistic we also sample binary random variables $\Phi_j \in \{0, 1\}$ to indicate whether the user knows the value of slot j or not. The agent maintains M multinomial distributions for its belief over user-goals given user utterances U_1^t till turn t. A slot distribution $p_j^t(v)$ for $v \in V^j$ is the probability at turn t that the user constraint for slot j is v. The agent also maintains M binomials $q_j^t = \Pr(\Phi_j = 1)$ which denote the probability that the user knows the value of slot j.

We also assume that column values are independently distributed to each other. This is a strong assumption but it allows us to model the user goal for each slot independently, as opposed to modeling the user goal over KB entities directly. Typically $\max_j |V^j| < N$ and hence this assumption reduces the number of parameters in the belief tracker.

3.3 Soft-KB Lookup

Let $p_{\mathcal{T}}^t(i) = \Pr(G = i|U_1^t)$ be the posterior probability that the user is interested in row i of the table, given the utterances up to turn t. We assume all probabilities are conditioned on user inputs U_1^t and drop it from the notation below. From our assumption of independence of slot values:

$$p_{\mathcal{T}}^t(i) \propto \prod_{j=1}^{M} \Pr(G_j = i), \qquad (1)$$

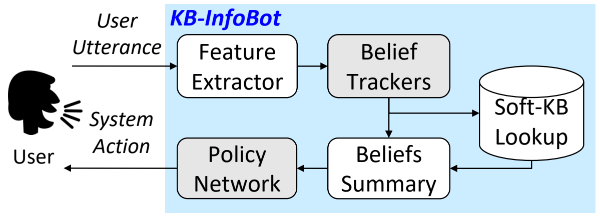


Figure 3: High-level overview of the end-to-end KB-InfoBot. Components with trainable parameters are highlighted in gray.

where $\Pr(G_j = i)$ denotes the posterior probability of user goal for slot j pointing to $\mathcal{T}_{i,j}$. We can marginalize this over Φ_j to get:

$$\begin{aligned} \Pr(G_j = i) &= \sum_{\phi=0}^{1} \Pr(G_j = i, \Phi_j = \phi) \qquad (2) \\ &= q_j^t \Pr(G_j = i|\Phi_j = 1) + \\ &\quad (1 - q_j^t) \Pr(G_j = i|\Phi_j = 0). \end{aligned}$$

For $\Phi_j = 0$, the user does not know the value of the slot, hence we assume a uniform prior over the rows of $\mathcal{T}$:

$$\Pr(G_j = i|\Phi_j = 0) = \frac{1}{N}, \quad 1 \leq i \leq N \qquad (3)$$

For $\Phi_j = 1$, the user knows the value of slot j, but this may be missing from $\mathcal{T}$, and we again have two cases:

$$\Pr(G_j = i|\Phi_j = 1) = \begin{cases} \frac{1}{N}, & i \in M_j \\ \frac{p_j^t(v)}{N_j(v)} \left(1 - \frac{|M_j|}{N}\right), & i \notin M_j \end{cases} \qquad (4)$$

Here, $p_j^t(v)$ is the slot distribution from the belief tracker, and $N_j(v)$ is the count of value v in slot j. Detailed derivation for (4) is provided in the appendix. Combining (1), (2), (3), and (4) gives us the procedure for computing the posterior over KB entities.

4 End-to-End KB-InfoBot

Figure 3 shows an overview of the end-to-end KB-InfoBot. At each turn, the agent receives a natural language utterance u^t as input, and selects an action a^t as output. The action space, denoted by $\mathcal{A}$, consists of $M + 1$ actions — *request(slot=i)* for $1 \leq i \leq M$ will ask the user for the value of slot i, and *inform(I)* will inform the user with an ordered list of results I from the KB. The dialogue ends once the agent chooses *inform*. We describe each of the components in detail below.

Feature Extractor: The feature extractor converts user input u^t into a vector representation x^t. In our implementation we use a simple bag of n-grams (with $n = 2$) representation, where each element of x^t is an integer indicating the count of a particular n-gram in u^t. We let V^n denote the number of unique n-grams, hence $x^t \in \mathbb{R}^{V^n}$. This module could potentially be replaced with a more sophisticated NLU unit, but for the user simulator we consider below the vocabulary size is relatively small ($V^n = 3078$), and doing so did not yield any improvements.

Belief Trackers: The KB-InfoBot consists of M belief trackers, one for each slot. Each tracker has input x^t and produces two outputs, p^t_j and q^t_j, which we shall collectively call the *belief state*: p^t_j is a multinomial distribution over the slot values v, and q^t_j is a scalar probability of the user knowing the value of that slot. It is common to use recurrent neural networks for belief tracking (Henderson et al., 2014; Wen et al., 2016b) since the output distribution at turn t depends on all user inputs till that turn. We use a Gated Recurrent Unit (GRU) (Cho et al., 2014) for each tracker, which, starting from $h^0_j = \mathbf{0}$ maintains a summary state h^t_j as follows:

$$\begin{aligned}
r^t_j &= \sigma(W^r_j x^t + U^r_j h^{t-1}_j + b^r) \\
z^t_j &= \sigma(W^z_j x^t + U^z_j h^{t-1}_j + b^z) \\
\tilde{h}^t_j &= \tanh(W^h_j x^t + U^h_j (r^t_j \cdot h^{t-1}_j) + b^h) \\
h^t_j &= (1 - z^t_j) \cdot h^{t-1}_j + z^t_j \cdot \tilde{h}^t_j \,. \qquad (5)
\end{aligned}$$

Here the subscript j and superscript t stand for the tracker index and dialogue turn respectively, and σ denotes the sigmoid nonlinearity. The output $h^t_j \in \mathbb{R}^d$ can be interpreted as a summary of what the user has said about slot j till turn t. The belief states are computed from this vector as follows:

$$p^t_j = \text{softmax}(W^p_j h^t_j + b^p_j) \qquad (6)$$

$$q^t_j = \sigma(W^\Phi_j h^t_j + b^\Phi_j) \qquad (7)$$

The key differences between the belief tracker described here and the one presented in (Wen et al., 2016b) are as follows: (1) we model the probability that user does not know the value of a slot separately, as opposed to treating it as a special value for the slot, since this is a very different type of object; (2) we use GRU units instead of a Jordan-type RNN, and use summary states h^t_j instead of tying together RNN weights; (3) we use n-gram features for simplicity instead of CNN features.

Soft-KB Lookup: This module uses the procedure described in section 3.3 to compute the posterior over the EC-KB $p^t_{\mathcal{T}} \in \mathbb{R}^N$ from the belief states described above. Note that this is a fixed, differentiable operation without any trainable parameters.

Collectively, outputs of the belief trackers and the soft-KB lookup can be viewed as the current dialogue state internal to the KB-InfoBot. Let $s^t = [p^t_1, p^t_2, ..., p^t_M, q^t_1, q^t_2, ..., q^t_M, p^t_{\mathcal{T}}]$ be the vector of size $\sum_j V^j + M + N$ denoting this state.

Beliefs Summary: At this stage it is possible for the agent to directly use the state vector s^t to select its next action a^t. However, the large size of the state vector would lead to a large number of parameters in the policy network. To improve efficiency we extract summary statistics from the belief states, similar to (Williams and Young, 2005; Gašić et al., 2009).

We summarize each slot into an entropy statistic over a distribution w^t_j computed from elements of the KB posterior $p^t_{\mathcal{T}}$ as follows:

$$w^t_j(v) \propto \sum_{i:\mathcal{T}_{i,j}=v} p^t_{\mathcal{T}}(i) + p^0_j \sum_{i:\mathcal{T}_{i,j}=\Psi} p^t_{\mathcal{T}}(i) \,. \qquad (8)$$

Here, p^0_j is a prior distribution over the values of slot j, estimated using counts of each value in the KB. Intuitively, the probability mass of v in this distribution is the agent's confidence that the user goal has value v in slot j. This confidence is a sum of two terms: (i) sum of KB posterior probabilities of rows which have value v, and (ii) sum of KB posterior probabilities of rows whose value is unknown, multiplied by the prior probability that an unknown might in fact be v. These two terms correspond to the two terms in (8) respectively. The summary statistic for slot j is then the entropy $H(w^t_j)$ of this weighted probability distribution. The KB posterior $p^t_{\mathcal{T}}$ is also summarized into an entropy statistic $H(p^t_{\mathcal{T}})$.

The scalar probabilities of the user knowing the value of a slot are passed as is to the policy network. Hence, the final summary vector which is input to the policy network is $\tilde{s}^t = [H(\tilde{p}_1^t), ..., H(\tilde{p}_M^t), q_1^t, ..., q_M^t, H(p_\mathcal{T}^t)]$. Note that this vector has size $2M + 1$.

Policy Network: The policy network's job is to select the next action based on the current summary state $\tilde{s}^t$ and the dialogue history. Similar to (Williams and Zweig, 2016; Zhao and Eskenazi, 2016), we use an RNN to allow the network to maintain an internal state of dialogue history. Specifically, we use a GRU unit (see eq 5) followed by a fully-connected layer and softmax nonlinearity to model the policy ($W^\pi \in \mathrm{R}^{|\mathcal{A}|\times d}$, $b^\pi \in \mathrm{R}^{|\mathcal{A}|}$):

$$h_\pi^t = \mathrm{GRU}(\tilde{s}^1, ..., \tilde{s}^t) \tag{9}$$

$$\pi = \mathrm{softmax}(W^\pi h_\pi^t + b^\pi)\,. \tag{10}$$

Action Selection: During the course of the dialogue, the agent samples its actions from the policy π. If this action is *inform()*, it must also provide an ordered set $I = (i_1, i_2, \ldots, i_R)$ of indices from the KB to the user. We assume a search-engine type setting where the agent returns a list of entities and the dialogue is considered a success if the correct entity is in top R items in the list. Since we want to learn the KB posterior $p_\mathcal{T}^t$ using reinforcement learning, we can view it as another policy, and sample results from the following distribution:

$$\mu(I) = p_\mathcal{T}^t(i_1) \times \frac{p_\mathcal{T}^t(i_2)}{1 - p_\mathcal{T}^t(i_1)} \times \cdots\,. \tag{11}$$

5 Training

5.1 Reinforcement Learning

The KB-InfoBot agent described above samples system actions from the policy π and KB results from the distribution μ. This allows the agent to explore both the space of actions $\mathcal{A}$ as well as the space of all possible KB results. This formulation leads to a modified version of the episodic REINFORCE algorithm (Williams, 1992) which we describe below.

We can write the expected discounted return of the agent under policy π as follows:

$$J(\theta) = \mathbf{E}\left[\sum_{h=0}^{H} \gamma^h r_h\right] \tag{12}$$

Here, the expectation is over all possible trajectories τ of the dialogue, θ denotes the parameters of the end-to-end system, H is the maximum length of an episode, γ is the discounting factor, and r_h the reward observed at turn h. We can use the likelihood ratio trick (Glynn, 1990) to write the gradient of the objective as follows:

$$\nabla_\theta J(\theta) = \mathbf{E}\left[\nabla_\theta \log p_\theta(\tau) \sum_{h=0}^{H} \gamma^h r_h\right], \tag{13}$$

where $p_\theta(\tau)$ is the probability of observing a particular trajectory under the current policy. With a Markovian assumption, we can write

$$p_\theta(\tau) = \left[p(s_0) \prod_{k=0}^{H} p(s_{k+1}|s_k, a_k)\pi_\theta(a_k|s_k)\right] \mu_\theta(I), \tag{14}$$

where θ denotes dependence on the neural network parameters. Notice the last term μ_θ above which is the posterior of a set of results from the KB. From 13,14 we obtain

$$\begin{aligned}\nabla_\theta J(\theta) =& \mathbf{E}_{a\sim\pi, I\sim\mu}\Big[\big(\nabla_\theta \log \mu_\theta(I) + \\ & \sum_{h=0}^{H} \nabla_\theta \log \pi_\theta(a_h)\big) \sum_{k=0}^{H} \gamma^k r_k\Big],\end{aligned} \tag{15}$$

where the expectation is now over all possible action sequences and the KB results, since gradient of the other terms in $p_\theta(\tau)$ is 0. This expectation is estimated using a mini-batch of dialogues of size B, and we use RMSProp (Hinton et al., 2012) updates to train the parameters θ.

5.2 Imitation Learning

In theory, both the belief trackers and policy network can be trained from scratch using only the reinforcement learning objective described above. In practice, however, for a moderately sized KB, the agent almost always fails if starting from random initialization. In this case, credit assignment is difficult for the agent, since it does not know whether the failure is due to an incorrect sequence of actions or incorrect set of results from the KB. Hence, at the beginning of training we have an imitation learning phase where the belief trackers and policy network are trained to mimic a simple hand-designed rule-based agent. The rule-based agent is described in detail in the next section. Here, we give the imitation learning objective used to bootstrap the KB-InfoBot.

Assume that $\hat{p}_j^t$ and $\hat{q}_j^t$ are the belief states from a rule-based agent, and $\hat{a}^t$ its action at turn t. Then the loss function in imitation learning is:

$$\mathcal{L}(\theta) = \mathbf{E}\big[D(\hat{p}_j^t||p_j^t(\theta)) + H(\hat{q}_j^t, q_j^t(\theta)) - \log \pi_\theta(\hat{a}^t)\big], \tag{16}$$

where $D(p||q)$ denotes the Kullback-Leibler divergence between p and q, and $H(p,q)$ denotes the cross-entropy between p and q. The last term is a standard supervised cross-entropy loss between the rule-based agent's action and its probability in the KB-InfoBot's policy. The expectation is estimated as an average over the minibatch, and we use stochastic gradient descent to optimize the loss.

5.3 User Simulator

To evaluate the performance of the KB-InfoBot, we make use of a rule-based stochastic simulated user. At the beginning of each dialogue, the simulated user randomly samples a target entity from the EC-KB and a random combination of *informable slots* for which it knows the value of the target. The remaining slot-values are unknown to the user. The user initiates the dialogue by providing a subset of its informable slots to the agent and requesting for an entity which matches them. In subsequent turns, if the agent requests for the value of a slot, the user complies by providing it or informs the agent that it does not know that value. If the agent informs results from the KB, the simulator checks whether the target is among them and provides the reward.

We convert dialogue acts from the user into natural language utterances using a separately trained natural language generator (NLG). The NLG is trained in a sequence-to-sequence fashion, using conversations between humans collected by crowd-sourcing. It takes the dialogue actions (DAs) as input, and generates template-like sentences with slot placeholders via an LSTM decoder. Then, a post-processing scan is performed to replace the slot placeholders with their actual values, which is similar to the decoder module in (Wen et al., 2015; Wen et al., 2016a). In the LSTM decoder, we apply beam search, which iteratively considers the top k best sentences up to time step t when generating the token of the time step $t+1$. For the sake of the trade-off between the speed and performance, we use the beam size of 3 in the following experiments.

There are several sources of error in user utterances. Any value provided by the user may be corrupted by noise, or substituted completely with an incorrect value of the same type (e.g., "Bill Murray" might become just "Bill" or "Tom Cruise"). The NLG described above is inherently stochastic, and may sometimes generate utterances irrelevant to the agent request. By increasing the temperature of the output softmax in the NLG we can increase the noise in user utterances.

6 Experiments and Results

6.1 Baselines

We compare our end-to-end model with two sets of baselines. **Rule-Based** agents consist of hand-designed belief trackers and a hand-designed policy. The belief trackers search for tokens in the user input which match a slot-value in the KB, and do a Bayesian update on the probability mass $\hat{p}_j^t(v)$ associated with the values found. If the agent asks for a slot, but does not find any value for that slot in the user response, then the corresponding *don't-care* probability $\hat{q}_j^t$ for that slot is set to 1. We compare three variants of the hand-designed policy, which differ in terms of the KB-lookup method. The *No-KB* version ignores the KB and selects its actions by always asking for the slot with maximum entropy $H(\hat{p}_j^t)$. The *Hard-KB* version performs a hard-KB lookup and selects the next action based on the entropy of the slots and the number of retrieved results. Finally, the *Soft-KB* version computes the full posterior over the KB and selects actions based on the summary statistics described in section 4. All these agents are variants of the EMDM strategy proposed in (Wu et al., 2015), with the difference being in the way the entropy is computed. At the end of the dialogue, all three agents inform the user with the top results from the KB posterior p_T^t, hence the difference only lies in the policy for action selection.

The second set of baselines, **Simple-RL** agents, retain the hand-designed belief trackers as described above, but use the GRU policy network described in section 4 instead of a hand-designed policy. We again compare three variants of these agents, which

Table 1: Movies-KB statistics. Total number of movies and unique values for each slot are given.

Relation Type	# Unique Values
Actor	51
Director	51
MPAA Rating	67
Critic Rating	68
Genre	21
Release Year	10
Movie	**428**

differ only in the inputs to the policy network. The *No-KB* version only takes entropy $H(\hat{p}_j^t)$ of each of the slot distributions. The *Hard-KB* version takes entropy of the slots along with a 6-bin one-hot encoding of the number of retrieved results (no match, 1 match, ... or more than 5 matches). This is the same approach as in (Wen et al., 2016b), except that we take entropy instead of summing probabilities. Lastly, The *Soft-KB* version takes the summary statistics of the slots and posterior over the KB described in section 4. In addition we also append the agent beliefs $\hat{q}_j^t$ whether the user knows the value of a slot, and a one-hot encoding of the previous agent action to the input for each of these versions. The policy network produces a distribution over the $M + 1$ valid actions available to the agent. The *inform* action is accompanied with results from the posterior p_T^t. During training actions are sampled from the output for exploration, but in evaluation actions are determined via *argmax*.

6.2 Movies-KB

In our experiments, we use a movie-centric knowledge base constructed using the IMDBPy[1] package. We selected a subset of movies released after 2007, and retained 6 slots. Statistics for this KB are given in Table 1. The original KB was modified to reduce the number of actors and directors in order to make the task more challenging[2]. We also randomly remove 20% of the values from the agent's copy of the KB to simulate a real-world scenario where the KB may be incomplete. The user, however, may still know these values.

[1]`http://imdbpy.sourceforge.net/`

[2]We restricted the vocabulary to the first few unique values of these slots and replaced all other values with a random value from this set

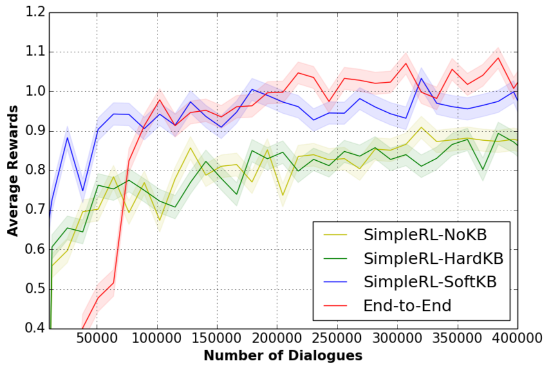


Figure 4: Average rewards and their std error during training for each of the models. Evaluation done at intervals of 100 updates, by choosing the optimal policy actions for 2000 simulations.

6.3 Hyperparameters

We use GRU hidden state size of $d = 50$ for the Simple-RL baselines and $d = 100$ for the end-to-end system, a learning rate of 0.05 for the imitation learning phase and 0.005 for the reinforcement learning phase, and minibatch size 128. Imitation learning was performed for 500 updates, after which the agent switched to reinforcement learning. The maximum length of a dialogue is limited to 10 turns,[3] beyond which the dialogue is deemed a failure. The input vocabulary is constructed from the NLG vocabulary and bigrams in the KB, and its size is 3078. The agent receives a positive reward if the user target is in top $R = 5$ results returned by it; this reward is computed as $2(1 - (r - 1)/R)$, where r is the actual rank of the target. For a failed dialogue the agent receives a reward of -1, and at each turn it receives a reward of -0.1 since we want it to complete the task in the shortest time possible. The discounting factor γ is set to 0.99.

6.4 Performance Comparison

We compare each of the discussed models along three metrics: the average rewards obtained, success rate (where success is defined as providing the user target among top R results), and the average number of turns per dialogue. Parameters of the rule-based baselines control the trade-off between average number of turns and success rate. We tuned these using grid-search and selected the combination with highest average reward.

[3]A turn consists of one user action and one agent action.

Table 2: Performance Comparison. Average and std error for 5000 runs after choosing the best model during training.

Agent	**KB Lookup**	**Success Rate**	**Avg Turns**	**Avg Reward**
Rule-based	No-KB	0.77 ± 0.01	5.05 ± 0.01	0.74 ± 0.02
Rule-based	Hard-KB	0.73 ± 0.01	3.65 ± 0.01	0.75 ± 0.02
Rule-based	Soft-KB	0.76 ± 0.01	3.94 ± 0.03	0.83 ± 0.02
Simple-RL	No-KB	0.76 ± 0.01	3.32 ± 0.02	0.87 ± 0.02
Simple-RL	Hard-KB	0.75 ± 0.01	$\mathbf{3.07 \pm 0.01}$	0.86 ± 0.02
Simple-RL	Soft-KB	0.80 ± 0.01	3.37 ± 0.03	0.98 ± 0.02
End2End-RL	Soft-KB	$\mathbf{0.83 \pm 0.01}$	3.27 ± 0.03	$\mathbf{1.10 \pm 0.02}$

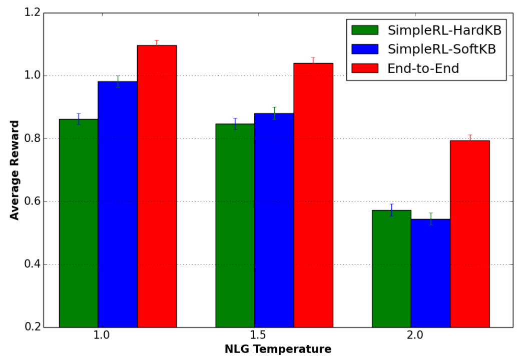


Figure 5: Variation in average rewards as temperature of softmax in NLG output is increased. Higher temperature leads to more noise in output. Average over 5000 simulations after selecting the best model during training.

Figure 4 shows how the reinforcement-learning agents perform as training progresses. This figure was generated by fixing the model every 100 updates, and performing 2000 simulations while selecting greedy policy actions. Table 2 shows the performance of each model over a further 5000 simulations, after selecting the best model during training, and selecting greedy policy actions.

The Soft-KB versions outperform both Hard-KB and No-KB counterparts, which perform similarly, in terms of average reward. The benefit comes from achieving a similar or higher success rate in reduced number of turns. Note that all baseline agents share the same belief trackers, but by re-asking values of some slots they can have different posteriors $p_{\mathcal{T}}^t$ to inform the results. Having full information about the current state of beliefs over the KB helps the Soft-KB agent discover better policies. Further, reinforcement learning helps discover better policies than the hand-crafted rule-based agents, and hence Simple-RL agents outperform the Rule-Based agents. All baseline agents, however, are limited by the rule-based belief trackers which remain fixed during training. The end-to-end agent is not limited as such, and is able to achieve a higher success rate and a higher average reward. This validates our motivation for introducing the Soft-KB lookup — the agent is able to improve both the belief trackers and policy network from user feedback directly.

Figure 5 shows the average reward of three of the agents as the temperature of the output softmax in the user simulator NLG is increased. A higher temperature means a more uniform output distribution, which leads to generic user responses irrelevant to the agent questions. This is a simple way of introducing noise in user responses. The performance of all three agents drops as the temperature is increased, but less so for the end-to-end agent, which can adapt its belief tracker to the inputs it receives.

7 Conclusion

We have presented an end-to-end differentiable dialogue agent for multi-turn information access. All components of the agent are trained using reinforcement learning from user feedback, by optimizing a modified version of the episodic REINFORCE objective. We have shown that starting from an imitation learning phase where the agent learns to mimic rule-based belief trackers and policy, the agent can successfully improve on its own through reinforcement learning. The gain in performance is especially high when the noise in user inputs is high.

A KB-InfoBot is a specific type of goal-oriented dialogue agent. Future work should focus on extending the techniques described here to other more general dialogue agents, such as a restaurant reservation agent or flight booking agent. We have also ignored scalability issues in this work, which is important for real-world sized knowledge bases and is a direction for future research.

References

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Antoine Bordes and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*.

Yun-Nung Chen, Dilek Hakkani-Tür, Gokhan Tur, Jianfeng Gao, and Li Deng. 2016. End-to-end memory networks with knowledge carryover for multi-turn spoken language understanding. In *Proceedings of The 17th Annual Meeting of the International Speech Communication Association*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*.

William W Cohen. 2016. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*.

Heriberto Cuayáhuitl. 2016. Simpleds: A simple deep reinforcement learning dialogue system. *International Workshop on Spoken Dialogue Systems (IWSDS)*.

Milica Gašić, Fabrice Lefevre, Filip Jurcicek, Simon Keizer, Francois Mairesse, Blaise Thomson, Kai Yu, Steve Young, et al. 2009. Back-off action selection in summary space-based POMDP dialogue systems. In *Automatic Speech Recognition & Understanding, 2009. ASRU 2009. IEEE Workshop on*, pages 456–461. IEEE.

Peter W Glynn. 1990. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84.

Dilek Hakkani-Tür, Gokhan Tur, Asli Celikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. 2016. Multi-domain joint semantic frame parsing using bi-directional RNN-LSTM. In *Proceedings of The 17th Annual Meeting of the International Speech Communication Association*.

Matthew Henderson, Blaise Thomson, and Steve Young. 2014. Word-based dialog state tracking with recurrent neural networks. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 292–299.

Matthew Henderson. 2015. Machine learning for dialog state tracking: A review. *Machine Learning in Spoken Language Processing Workshop*.

Geoffrey Hinton, N Srivastava, and Kevin Swersky. 2012. Lecture 6a overview of mini–batch gradient descent. *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online*.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation. *EMNLP*.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *EMNLP*.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. 2016a. Conditional generation and snapshot learning in neural dialogue systems. *EMNLP*.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. 2016b. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*.

Jason D Williams and Steve Young. 2005. Scaling up POMDPs for dialog management: The "Summary POMDP" method. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.*, pages 177–182. IEEE.

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

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Ji Wu, Miao Li, and Chin-Hui Lee. 2015. A probabilistic framework for representing dialog systems and entropy-based dialog management through dynamic stochastic state evolution. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(11):2026–2035.

Kaisheng Yao, Baolin Peng, Yu Zhang, Dong Yu, Geoffrey Zweig, and Yangyang Shi. 2014. Spoken language understanding using long short-term memory neural networks. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 189–194. IEEE.

Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li. 2016a. Neural generative question answering. *International Joint Conference on Artificial Intelligence*.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2016b. Neural enquirer: Learning to query tables. *International Joint Conference on Artificial Intelligence*.

Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. 2013. POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179.

Wojciech Zaremba and Ilya Sutskever. 2015. Reinforcement learning neural Turing machines-revised. *arXiv preprint arXiv:1505.00521*.

Tiancheng Zhao and Maxine Eskenazi. 2016. Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. *arXiv preprint arXiv:1606.02560*.

Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. 2013. Do we need entity-centric knowledge bases for entity disambiguation? In *Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies*, page 4. ACM.

Table 3: Sample dialogues between user simulator and SimpleRL-SoftKB and End2End-RL agents. At the end of each dialogue, the agent informs top 5 results from the KB posterior. User target, if informed, is in bold.

SimpleRL-SoftKB	**End-to-End**
User: I would like to know a movie whose critic_rating is 5.5 and actor is fifer.	
Agent: *request*(actor)	Agent: *request*(actor)
User: chad fifer	User: chad fifer
Agent: *request*(mpaa_rating)	Agent: *request*(critic_rating)
User: i cannot remember	User: 5.2
Agent: *request*(critic_rating)	Agent: *request*(mpaa_rating)
User: i think it is 5.2	User: i cannot remember
Agent: *request*(genre)	Agent: *request*(critic_rating)
User: i cannot remember	User: i think it is 5.5
Agent: *inform*(extranjera, promise, bling, the neighbor, lesson no. one)	Agent: *inform*(**le prince de ce monde**, tel pre telle fille, the neighbor, promise, krakatoa)
User: Which movie has sabine as director and comedy as genre?	
Agent: *request*(actor)	Agent: *request*(critic_rating)
User: i cannot remember	User: i think it is 6.9.
Agent: *request*(mpaa_rating)	Agent: *inform*(**ruddy hell! it's harry and paul**, made in america, rome-old and juli-eh, bathtime, numb)
User: i cannot remember	
Agent: *request*(critic_rating)	
User: i think it is 6.9	
Agent: *request*(critic_rating)	
User: i think it is 7.2	
Agent: *inform*(**ruddy hell! it's harry and paul**, homerazzi gotthard lange, bathtime, rome-old and juli-eh, numb)	

A Sample Dialogues

Table 3 shows some sample dialogues between the user simulator and SimpleRL-SoftKB and End2End-RL agents. User utterances are generated using the NLG described in text. Value of the *critic_rating* slot is a common source of error in the user simulator, and hence all learned policies tend to ask for this value multiple times.

B Posterior Derivation

Here, we present a derivation for equation 4, i.e., the posterior over the KB slot when the user knows the value of that slot. For brevity, we drop $\Phi_j = 0$ from the condition in all probabilities below. For the case when $i \in M_j$, we can write:

$$\begin{aligned} &\Pr(G_j = i) \\ &= \Pr(G_j \in M_j) \Pr(G_j = i | G_j \in M_j) \\ &= \frac{|M_j|}{N} \frac{1}{|M_j|} = \frac{1}{N}, \end{aligned} \tag{17}$$

where we assume all missing values to be equally likely, and estimate the prior probability of the goal being missing from the count of missing values in that slot. For the case when $i = v \notin M_j$:

$$\begin{aligned} &\Pr(G_j = i) \\ &= \Pr(G_j \notin M_j) \Pr(G_j = i | G_j \notin M_j) \\ &= \left(1 - \frac{|M_j|}{N}\right) \times \frac{p_j^t(v)}{N_j(v)}, \end{aligned} \tag{18}$$

where the second term comes from taking the probability mass associated with v in the belief tracker and dividing it equally among all rows with value v.