Flexibly-Structured Model for Task-Oriented Dialogues

Lei Shu[1] *, **Piero Molino**[2] , **Mahdi Namazifar**[2] , **Hu Xu**[1] , **Bing Liu**[1] , **Huaixiu Zheng**[2] , **Gokhan Tur**[2]
[1]Department of Computer Science, University of Illinois at Chicago
[2]Uber AI,
[1]{lshu3, hxu48, liub}@uic.edu,
[2]{piero, mahdin, huaixiu.zheng, gokhan}@uber.com

Abstract

This paper proposes a novel end-to-end architecture for task-oriented dialogue systems. It is based on a simple and practical yet very effective sequence-to-sequence approach, where language understanding and state tracking tasks are modeled jointly with a structured copy-augmented sequential decoder and a multi-label decoder for each slot. The policy engine and language generation tasks are modeled jointly following that. The copy-augmented sequential decoder deals with new or unknown values in the conversation, while the multi-label decoder combined with the sequential decoder ensures the explicit assignment of values to slots. On the generation part, slot binary classifiers are used to improve performance. This architecture is scalable to real-world scenarios and is shown through an empirical evaluation to achieve state-of-the-art performance on both the Cambridge Restaurant dataset and the Stanford in-car assistant dataset[1].

1 Introduction

A traditional task-oriented dialogue system is often composed of a few modules, such as natural language understanding, dialogue state tracking, knowledge base (KB) query, dialogue policy engine and response generation. Language understanding aims to convert the input to some predefined semantic frame. State tracking is a critical component that models explicitly the input semantic frame and the dialogue history for producing KB queries. The semantic frame and the corresponding belief state are defined in terms of informable slots values and requestable slots. Informable slot values capture information provided by the user so far, e.g., {*price=cheap, food=italian*} indicating the user wants a cheap Italian restaurant at this stage. Requestable slots capture the information requested by the user, e.g., {*address, phone*} means the user wants to know the address and phone number of a restaurant. Dialogue policy model decides on the system action which is then realized by a language generation component.

To mitigate the problems with such a classic modularized dialogue system, such as the error propagation between modules, the cascade effect that the updates of the modules have and the expensiveness of annotation, end-to-end training of dialogue systems was recently proposed (Liu and Lane, 2018; Williams et al., 2017; Lowe et al., 2017; Li et al., 2018; Liu et al., 2018; Budzianowski et al., 2018; Bordes et al., 2017; Wen et al., 2017b; Serban et al., 2016, among others). These systems train one whole model to read the current user's utterance, the past state (that may contain all previous interactions) and generate the current state and response.

There are two main approaches for modeling the belief state in end-to-end task-oriented dialogue systems in the literature: the *fully structured* approach based on classification (Wen et al., 2017b,a), and the *free-form* approach based on text generation (Lei et al., 2018). The fully structured approaches (Ramadan et al., 2018; Ren et al., 2018) use the full structure of the KB, both its schema and the values available in it, and assumes that the sets of informable slot values and requestable slots are fixed. In real-world scenarios, this assumption is too restrictive as the content of the KB may change and users' utterances may contain information outside the pre-defined sets. An ideal end-to-end architecture for state tracking should be able to identify the values of the informable slots and the requestable slots, easily adapt to new domains, to the changes in the content of the KB, and to the

*Work mostly performed as an intern at Uber AI Labs
[1]The code is available at https://github.com/uber-research/FSDM

Proceedings of the SIGDial 2019 Conference, pages 178–187
Stockholm, Sweden, 11-13 September 2019.

occurrence of words in users' utterances that are not present in the KB at training time, while at the same time providing the right amount of inductive bias to allow generalization.

Recently, a free-form approach called TSCP (Two Stage Copy Net) (Lei et al., 2018) was proposed. This approach does not integrate any information about the KB in the model architecture. It has the advantage of being readily adaptable to new domains and changes in the content of the KB as well as solving the out-of-vocabulary word problem by generating or copying the relevant piece of text from the user's utterances in its response generation. However, TSCP can produce invalid states (see Section 4). Furthermore, by putting all slots together into a sequence, it introduces an unwanted (artificial) order between different slots since they are encoded and decoded sequentially. It could be even worse if two slots have overlapping values, like departure and arrival airport in a travel booking system. As such, the unnecessary order of the slots makes getting rid of the invalid states a great challenge for the sequential decoder. As a summary, both approaches to state tracking have their weaknesses when applied to real-world applications.

This paper proposes the Flexibly-Structured Dialogue Model (FSDM) as a new end-to-end task-oriented dialogue system. The state tracking component of FSDM has the advantages of both fully structured and free-form approaches while addressing their shortcomings. On one hand, it is still structured, as it incorporates information about slots in KB schema; on the other hand, it is flexible, as it does not use information about the values contained in the KB records. This makes it easily adaptable to new values. These desirable properties are achieved by a separate decoder for each informable slot and a multi-label classifier for the requestable slots. Those components explicitly assign values to slots like the fully structured approach, while also preserving the capability of dealing with out-of-vocabulary words like the free-form approach. By using these two types of decoders, FSDM produces only valid belief states, overcoming the limitations of the free-form approach. Further, FSDM has a new module called response slot binary classifier that adds extra supervision to generate the slots that will be present in the response more precisely before generating the final textual agent response (see Section 3 for details).

The main contributions of this work are

1. FSDM, a task-oriented dialogue system with a new belief state tracking architecture that overcomes the limits of existing approaches and scales to real-world settings;

2. a new module, namely the response slot binary classifier, that helps to improve the performance of agent response generation;

3. FSDM achieves state-of-the-art results on both the Cambridge Restaurant dataset (Wen et al., 2017b) and the Stanford in-car assistant dataset (Eric et al., 2017) without the need for fine-tuning through reinforcement learning

2 Related Work

Our work is related to end-to-end task-oriented dialogue systems in general (Liu and Lane, 2018; Williams et al., 2017; Lowe et al., 2017; Li et al., 2018; Liu et al., 2018; Budzianowski et al., 2018; Bordes et al., 2017; Hori et al., 2016; Wen et al., 2017b; Serban et al., 2016, among others) and those that extend the Seq2Seq (Sutskever et al., 2014) architecture in particular (Eric et al., 2017; Fung et al., 2018; Wen et al., 2018). Belief tracking, which is necessary to form KB queries, is not explicitly performed in the latter works. To compensate, Eric et al. (2017); Xu and Hu (2018a); Wen et al. (2018) adopt a copy mechanism that allows copying information retrieved from the KB to the generated response. Fung et al. (2018) adopt Memory Networks (Sukhbaatar et al., 2015) to memorize the retrieved KB entities and words appearing in the dialogue history. These models scale linearly with the size of the KB and need to be retrained at each update of the KB. Both issues make these approaches less practical in real-world applications.

Our work is also akin to modularly connected end-to-end trainable networks (Wen et al., 2017b,a; Liu and Lane, 2018; Liu et al., 2018; Li et al., 2018; Zhong et al., 2018). Wen et al. (2017b) includes belief state tracking and has two phases in training: the first phase uses belief state supervision, and then the second phase uses response generation supervision. Wen et al. (2017a) improves Wen et al. (2017b) by adding a policy network using latent representations so that the dialogue system can be continuously improved through reinforcement learning. These methods utilize classification as a way to decode the belief state.

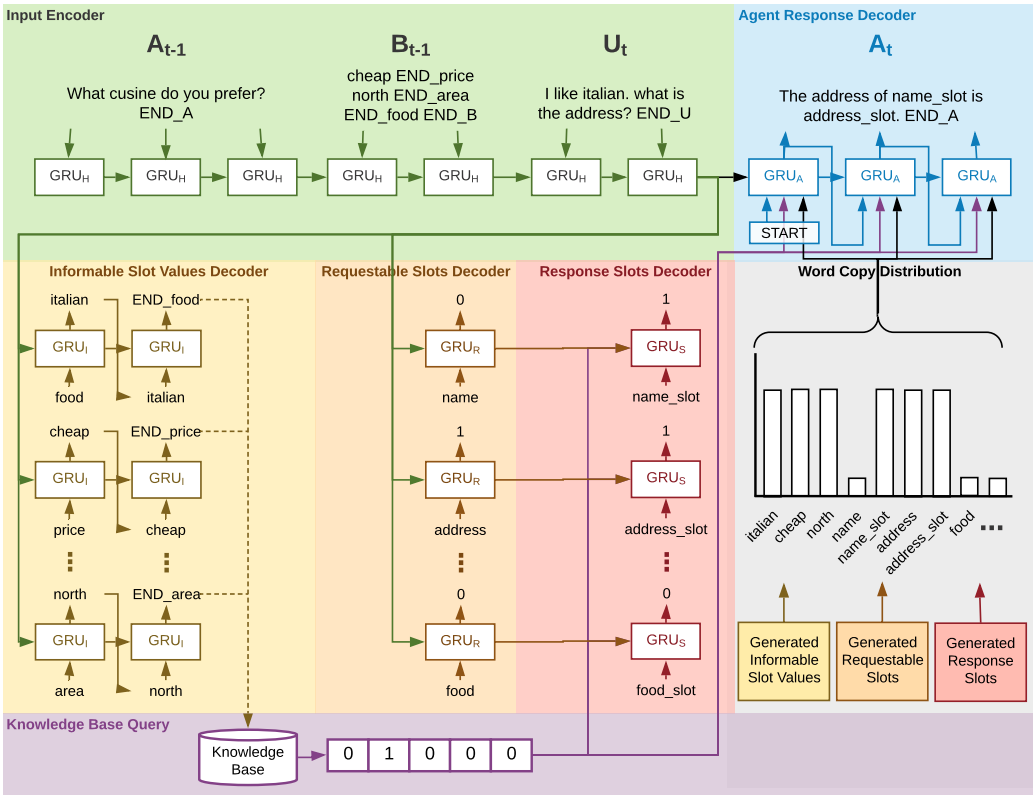

Figure 1: FSDM architecture illustrated by a dialogue turn from the Cambridge Restaurant dataset with the following components: an input encoder (green), a belief state tracker (yellow for the informable slot values, orange for the requestable slots), a KB query component (purple), a response slot classifier (red), a component that calculates word copy probability (grey) and a response decoder (blue). Attention connections are not drawn for brevity.

Lei et al. (2018) decode the belief state as well as the response in a free-form fashion, but it tracks the informable slot values without an explicit assignment to an informable slot. Moreover, the arbitrary order in which informable slot values and requestable slots are encoded and decoded suggests that the sequential inductive bias the architecture provides may not be the right one.

Other works (Jang et al., 2016; Henderson et al., 2014; Bapna et al., 2017; Kobayashi et al., 2018; Xu and Hu, 2018b) focus on the scalability of DST to large or changing vocabularies. Rastogi et al. (2017) score a dynamically defined set of candidates as informable slot values. Dernoncourt et al. (2016) addresses the problem of large vocabularies with a mix of rules and machine-learned classifiers.

3 Methodology

We propose a fully-fledged task-oriented dialogue system called Flexibly-Structured Dialogue Model (FSDM), which operates at the turn level. Its overall architecture is shown in Figure 1, which illustrates one dialogue turn. Without loss of generality, let us assume that we are on the t-th turn of a dialogue. FSDM has three (3) inputs: agent response and belief state of the $t-1$-th turn, and user utterance of the t-th turn. It has two (2) outputs: the belief state for the t-th turn that is used to query the KB, and the agent response of the t-th turn based on the query result. As we can see, belief tracking is the key component that turns unstructured user utterance and the dialogue history into a KB-friendly belief state. The success of retrieving the correct KB result and further generating the correct response to complete a task relies on the quality of the produced belief state.

FSDM contains five (5) components that work together in an end-to-end manner as follows: (1) The input is encoded and the last hidden state of the encoder serves as the initial hidden state of the belief state tracker and the response decoder; (2) Then, the belief state tracker generates a belief state $B_t = \{I_t, R_t\}$, where I_t is the set of constraints used for the KB query generated by the informable slots value decoder and R_t is the user requested slots identified by the requestable slots multi-label classifier; (3) Given I_t, the KB query component queries the KB and encodes the number of records

returned in a one-hot vector d_t; (4) The response slot binary classifier predicts which slots should appear in the agent response S_t; (5) Finally, the agent response decoder takes in the KB output d_t, a word copy probability vector $\mathcal{P}^c$ computed from I_t, R_t, S_t together with an attention on hidden states of the input encoder and the belief decoders, and generates a response A_t.

3.1 Input Encoder

The input contains three parts: (1) the agent response A_{t-1}, (2) the belief state B_{t-1} from the $(t-1)$-th turn and (3) the current user utterance U_t. These parts are all text-based and concatenated, and then consumed by the input encoder. Specifically, the belief state B_{t-1} is represented as a sequence of informable slot names with their respective values and requestable slot names. As an example, the sequence ⟨*cheap, end_price, italian, end_food, address, phone, end_belief*⟩ indicates a state where the user informed cheap and Italian as KB query constraints and requested the address and phone number.

The input encoder consists of an embedding layer followed by a recurrent layer with Gated Recurrent Units (GRU) (Cho et al., 2014). It maps the input $A_{t-1} \circ B_{t-1} \circ U_t$ (where $\circ$ denotes concatenation) to a sequence of hidden vectors $\{h_i^E | i = 1, \ldots, |A_{t-1} \circ B_{t-1} \circ U_t|\}$ so that $h_i^E = \text{GRU}_H(e^{A_{t-1} \circ B_{t-1} \circ U_t})$ where e is the embedding function that maps from words to vectors. The output of the input encoder is its last hidden state h_l^E, which is served as the initial state for the belief state and response decoders as discussed next.

3.2 Informable Slot Value Decoder

The belief state is composed of informable slot values I_t and the requestable slots R_t. We describe the generation of the former in this subsection and the latter in the next subsection.

The informable slot values track information provided by the user and are used to query the KB. We allow each informable slot to have its own decoder to resolve the unwanted artificial dependencies among slot values introduced by TSCP (Lei et al., 2018). As an example of artificial dependency, 'italian; expensive' appears a lot in the training data. During testing, even when the gold informable value is 'italian; moderate', the decoder may still generate 'italian; expensive'. Modeling one decoder for each slot exactly associates the values with the corresponding informable slot.

The informable slot value decoder consists of GRU recurrent layers with a copy mechanism as shown in the yellow section of Figure 1. It is composed of weight-tied GRU generators that take the same initial hidden state h_l^E, but have different start-of-sentence symbols for each unique informable slot. This way, each informable slot value decoder is dependent on the encoder's output, but it is also independent of the values generated for the other slots. Let $\{k^I\}$ denote the set of informable slots. The probability of the j^{th} word $P(y_j^{k^I})$ being generated for the slot k^I is calculated as follows: (1) calculate the attention with respect to the input encoded vectors to obtain the context vector $c_j^{k^I}$, (2) calculate the generation score $\phi_g(y_j^{k^I})$ and the copy score $\phi_c(y_j^{k^I})$ based on the current step's hidden state $h_j^{k^I}$, (3) calculate the probability using the copy mechanism:

$$
\begin{aligned}
c_j^{k^I} &= \text{Attn}(h_{j-1}^{k^I}, \{h_i^E\}), \\
h_j^{k^I} &= \text{GRU}_I\Big((c_j^{k^I} \circ e^{y_j^{k^I}}), h_{j-1}^{k^I}\Big), \\
\phi_g(y_j^{k^I}) &= W_g^{K^I} \cdot h_j^{k^I}, \\
\phi_c(y_j^{k^I}) &= \tanh(W_c^{K^I} \cdot h^{y_j^{k^I}}) \cdot h_j^{k^I}, \\
y_j^{k^I} &\in A_{t-1} \cup B_{t-1} \cup U_t, \\
P(y_j^{k^I} | y_{j-1}^{k^I}, h_{j-1}^{k^I}) &= \text{Copy}\Big(\phi_c(y_j^{k^I}), \phi_g(y_j^{k^I})\Big),
\end{aligned}
\tag{1}
$$

where for each informable slot k^I, $y_0^{k^I} = k^I$ and $h_0^{k^I} = h_l^E$, $e^{y_j^{k^I}}$ is the embedding of the current input word (the one generated at the previous step), and $W_g^{K^I}$ and $W_c^{K^I}$ are learned weight matrices. We follow (Gu et al., 2016) and (Bahdanau et al., 2015) for the copy $\text{Copy}(\cdot, \cdot)$ and attention $\text{Attn}(\cdot, \cdot)$ mechanisms implementation respectively.

The loss for the informable slot values decoder is calculated as follows:

$$
\begin{aligned}
\mathcal{L}^I = -\frac{1}{|\{k^I\}|} \frac{1}{|Y^{k^I}|} \sum_{k^I} \sum_j \\
\log P(y_j^{k^I} = z_j^{k^I} | y_{j-1}^{k^I}, h_{j-1}^{k^I}),
\end{aligned}
\tag{2}
$$

where Y^{K^I} is the sequence of informable slot value decoder predictions and z is the ground truth label.

3.3 Requestable Slot Binary Classifier

As the other part of a belief state, requestable slots are the attributes of KB entries that are explicitly requested by the user. We introduce a separate

multi-label requestable slots classifier to perform binary classification for each slot. This greatly resolves the issues of TSCP that uses a single decoder with each step having unconstrained vocabulary-size choices, which may potentially lead to generating non-slot words. Similar to the informable slots decoders, such a separate classifier also eliminates the undesired dependencies among slots.

Let $\{k^R\}$ denote the set of requestable slots. A single GRU cell is used to perform the classification. The initial state h_l^E is used to pay attention to the input encoder hidden vectors to compute a context vector c^{k^R}. The concatenation of c^{k^R} and e^{k^R}, the embedding vector of one requestable slot k^R, is passed as input and h_l^E as the initial state to the GRU. Finally, a sigmoid non-linearity is applied to the product of a weight vector W_y^R and the output of the GRU h^{k^R} to obtain y^{k^R}, which is the probability of the slot being requested by the user.

$$\begin{aligned} c^{k^R} &= \text{Attn}(h_l^E, \{h_i^E\}), \\ h^{k^R} &= \text{GRU}_R\Big((c^{k^R} \circ e^{k^R}), h_l^E\Big), \\ y^{k^R} &= \sigma(W_y^R \cdot h^{k^R}). \end{aligned} \tag{3}$$

The loss function for all requestable slot binary classifiers is:

$$\begin{aligned} \mathcal{L}^R = &- \frac{1}{|\{k^R\}|} \sum_{k^R} \\ & z^{k^R} \log(y^{k^R}) + (1 - z^{k^R}) \log(1 - y^{k^R}). \end{aligned} \tag{4}$$

3.4 Knowledge Base Query

The generated informable slot values $I_t = \{Y^{k^I}\}$ are used as constraints of the KB query. The KB is composed of one or more relational tables and each entity is a record in one table. The query is performed to select a subset of the entities that satisfy those constraints. For instance, if the informable slots are {*price=cheap, area=north*}, all the restaurants that have attributes of those fields equal to those values will be returned. The output of this component, the one-hot vector d_t, indicates the number of records satisfying the constraints. d_t is a five-dimensional one-hot vector, where the first four dimensions represent integers from 0 to 3 and the last dimension represents 4 or more matched records. It is later used to inform the response slot binary classifier and the agent response decoder.

3.5 Response Slot Binary Classifier

In order to incorporate all the relevant information about the retrieved entities into the response, FSDM introduces a new response slot binary classifier. Its inputs are requestable slots and KB queried result d_t and the outputs are the response slots to appear in the agent response. Response slots are the slot names that are expected to appear in a de-lexicalized response (discussed in the next subsection). For instance, assume the requestable slot in the belief state is "address" and the KB query returned one candidate record. The response slot binary classifier may predict name_slot, address_slot and area_slot, which are expected to appear in an agent response as "name_slot is located in address_slot in the area_slot part of town"[2].

The response slots $\{k^S\}$ map one-to-one to the requestable slots $\{k^R\}$. The initial state of each response slot decoder is the last hidden state of the corresponding requestable slot decoder. In this case, the context vector c^{k^S} is obtained by paying attention to all hidden vectors in the informable slot value decoders and requestable slots classifiers. Then, the concatenation of the context vector c^{k^S}, the embedding vector of the response slot e^{k^S} and the KB query vector d_t are used as input to a single GRU cell. Finally, a sigmoid non-linearity is applied to the product of a weight vector W_y^S and the output of the GRU h^{k^S} to obtain a probability y^{k^S} for each slot that is going to appear in the answer.

$$\begin{aligned} c^{k^S} &= \text{Attn}(h^{k^R}, \\ &\{h_i^{k^I} | k^I \in K^I, i \leq |Y^{k^I}|\} \cup \{h^{k^R} | k^R \in K^R\}), \\ h^{k^S} &= \text{GRU}_S\Big((c^{k^S} \circ e^{k^S} \circ d_t), h^{k^R}\Big), \\ y^{k^S} &= \sigma(W_y^S \cdot h^{k^S}). \end{aligned} \tag{5}$$

The loss function for all response slot binary classifiers is:

$$\begin{aligned} \mathcal{L}^S = &- \frac{1}{|\{k^S\}|} \sum_{k^S} \\ & z^{k^S} \log(y^{k^S}) + (1 - z^{k^S}) \log(1 - y^{k^S}). \end{aligned} \tag{6}$$

3.6 Word Copy Probability and Agent Response Decoder

Lastly, we introduce the agent response decoder. It takes in the generated informable slot values, requestable slots, response slots, and KB query result and generates a (de-lexicalized) response. We adopt a copy-augmented decoder (Gu et al., 2016) as architecture. The canonical copy mechanism only takes a sequence of word indexes as inputs but

[2] Before the agent response is presented to the user, those slot names are replaced by the actual values of the KB entries.

does not accept the multiple Bernoulli distributions we obtain from sigmoid functions. For this reason, we introduce a vector of independent word copy probabilities $\mathcal{P}^C$, which is constructed as follows:

$$\mathcal{P}^{\mathcal{C}}(w) = \begin{cases} y^{k^R}, & \text{if } w = k^R, \\ y^{k^S}, & \text{if } w = k^S, \\ 1, & \text{if } w \in I_t, \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

where if a word w is a requestable slot or a response slot, the probability is equal to their binary classifier output; if a word appears in the generated informable slot values, its probability is equal to 1; for the other words in the vocabulary the probability is equal to 0. This vector is used in conjunction with the agent response decoder prediction probability to generate the response.

The agent response decoder is responsible for generating a de-lexicalized agent response. The response slots are substituted with the values of the results obtained by querying the KB before the response is returned to the user.

Like the informable slot value decoder, the agent response decoder also uses a copy mechanism, so it has a copy probability and generation probability. Consider the generation of the j^{th} word. Its generation score ϕ_g is calculated as:

$$\begin{aligned} c_j^{A^E} &= \text{Attn}(h_{j-1}^A, \{h_i^E\}), \\ c_j^{A^B} &= \text{Attn}(h_{j-1}^A, \{h_i^{k^I} | k^I \in K^I, i \leq |Y^{k^I}|\} \\ &\cup \{h^{k^R} | k^R \in K^R\}) \cup \{h^{k^S} | k^S \in K^S\}), \\ h_j^A &= \text{GRU}_A\Big((c_j^{A^E} \circ c_j^{A^B} \circ e_j^A \circ d_t), h_{j-1}^A\Big), \\ \phi_g(y_j^A) &= W_g^A \cdot h_j^A, \end{aligned} \tag{8}$$

where $c_j^{A^E}$ is a context vector obtained by attending to the hidden vectors of the input encoder, $c_j^{A^B}$ is a context vector obtained by attending to all hidden vectors of the informable slot value decoder, requestable slot classifier and response slot classifier, and W_g^A is a learned weight matrix. The concatenation of the two context vectors $c_j^{A^E}$ and $c_j^{A^B}$, the embedding vector e_j^A of the previously generated word and the KB query output vector d_t is used as input to a GRU. Note that the initial hidden state is $h_0^A = h_l^E$. The copy score ϕ_c is calculated as:

$$\phi_c(y_j^A) = \begin{cases} \mathcal{P}^C(y_j^A) \cdot \tanh(W_c^A \cdot h^{y_j^A}) \cdot h_j^A, \\ \text{if } y_j^A \in I_t \cup K^R \cup K^S, \\ \mathcal{P}^C(y_j^A), \text{otherwise}, \end{cases} \tag{9}$$

where W_c^A is a learned weight matrix. The final probability is:

$$P(y_j^A | y_{j-1}^A, h_{j-1}^A) = \text{Copy}(\phi_g(y_j^A), \phi_c(y_j^A)). \tag{10}$$

Let z denote the ground truth de-lexicalized agent response. The loss for the agent response decoder is calculated as follows where Y^A is the sequence of agent response decoder prediction:

$$\mathcal{L}^A = -\frac{1}{|Y^A|} \sum_j \log P(y_j^A = z_j^A | y_{j-1}^A, h_{j-1}^A). \tag{11}$$

CamRest: restaurant reservation			
dialogue split	train: 408	dev: 136	test: 136
# of keys	informable: 3	requestable: 7	response: 7
database record	99		
KVRET: navigation, weather, calendar scheduling			
dialogue split	train: 2425	dev: 302	test: 302
# of keys	informable: 10	requestable: 12	response: 12
database record	284		

Table 1: Dataset

3.7 Loss Function

The loss function of the whole network is the sum of the four losses described so far for the informable slot values $\mathcal{L}^I$, requestable slot $\mathcal{L}^R$, response slot $\mathcal{L}^S$ and agent response decoders $\mathcal{L}^A$, weighted by α hyperparameters:

$$\mathcal{L} = \alpha^I \mathcal{L}^I + \alpha^R \mathcal{L}^R + \alpha^S \mathcal{L}^S + \alpha^A \mathcal{L}^A. \tag{12}$$

The loss is optimized in an end-to-end fashion, with all modules trained simultaneously with loss gradients back-propagated to their weights. In order to do so, ground truth results from database queries are also provided to the model to compute the d_t, while at prediction time results obtained by using the generated informable slot values I_t are used.

4 Experiments

We tested the FSDM on the Cambridge Restaurant dataset (CamRest) (Wen et al., 2017b) and the Stanford in-car assistant dataset (KVRET) (Eric et al., 2017) described in Table 1.

4.1 Preprocessing and Hyper-parameters

We use NLTK (Bird et al., 2009) to tokenize each sentence. The user utterances are precisely the original texts, while all agent responses are de-lexicalized as described in (Lei et al., 2018). We obtain the labels for the response slot decoder from the de-lexicalized response texts. We use 300-dimensional GloVe embeddings (Pennington et al., 2014) trained on 840B words. Tokens not present

Dataset	CamRest						KVRET					
Method	Inf P	Inf R	Inf F_1	Req P	Req R	Req F_1	Inf P	Inf R	Inf F_1	Req P	Req R	Req F_1
TSCP/RL†	0.970	0.971	0.971	0.983	0.935	0.959	**0.936**	0.874	0.904	0.725	0.485	0.581
TSCP†	0.970	0.971	0.971	0.983	0.938	0.960	0.934	0.890	0.912	0.701	0.435	0.526
FSDM/Res	0.979	0.984	0.978	0.994	0.947	0.967	0.918	0.930	0.925	0.812	0.993	0.893
FSDM	**0.983***	**0.986***	**0.984***	**0.997***	**0.952**	**0.974***	0.92	**0.935***	**0.927***	**0.819***	**1.000***	**0.900***

Table 2: Turn-level performance results. **Inf**: Informable, **Req**: Requestable, **P**: Precision, **R**: Recall. Results marked with † are computed using available code, and all the other ones are reported from the original papers. $*$ indicates the improvement is statistically significant with $p = 0.05$.

Dataset	CamRest			KVRET		
Method	BLEU	EMR	$SuccF_1$	BLEU	EMR	$SuccF_1$
NDM	0.212	0.904	0.832	0.186	0.724	0.741
LIDM	0.246	0.912	0.840	0.173	0.721	0.762
KVRN	0.134	-	-	0.184	0.459	0.540
TSCP	0.253	0.927	0.854	**0.219**	0.845	0.811
TSCP/RL †	0.237	0.915	0.826	0.195	0.809	0.814
TSCP†	0.237	0.913	0.841	0.189	0.833	0.81
FSDM/St	0.245	-	0.847	0.204	-	0.809
FSDM/Res	0.251	0.924	0.855	0.209	0.834	0.815
FSDM	**0.258***	**0.935***	**0.862***	0.215	**0.848***	**0.821***

Table 3: Dialogue level performance results. **$SuccF_1$**: Success F_1 score, **EMR**: Entity Match Rate. Results marked with † are computed using available code, and all the other ones are reported from the original papers. $*$ indicates the improvement is statistically significant with $p = 0.05$.

user msg	what is the date and time of my next meeting and who will be attending it ?
	belief state
GOLD	informable slot (event=meeting), requestable slot (date, time, party)
TSCP	'meeting' '⟨EOS_Z1⟩' 'date' ';' 'party'
FSDM	event=meeting date=True time=True party = True
	agent response
GOLD	your next meeting is with party_SLOT on the date_SLOT at time_SLOT.
TSCP	your next meeting is at time_SLOT on date_SLOT at time_SLOT .
FSDM	you have a meeting on date_SLOT at time_SLOT with party_SLOT

Table 4: Example of generated belief state and response for calendar scheduling domain

in GloVe are initialized to be the average of all other embeddings plus a small amount of random noise to make them different from each other.

We optimize both training and model hyperparameters by running Bayesian optimization over the product of validation set BLEU, EMR, and $SuccF_1$ using skopt[3]. The model that performed the best on the validation set uses Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.00025 for minimizing the loss in Equation 12 for both datasets. We apply dropout with a rate of 0.5 after the embedding layer, the GRU layer and any linear layer for CamRest and 0.2 for KVRET. The dimension of all hidden states is 128 for CamRest and 256 for KVRET. Loss weights α^I, α^R, α^S, α^A are 1.5, 9, 8, 0.5 respectively for CamRest and 1, 3, 2, 0.5 for KVRET.

[3] https://scikit-optimize.github.io/

4.2 Evaluation Metrics

We evaluate the performance concerning belief state tracking, response language quality, and task completion. For belief state tracking, we report precision, recall, and F_1 score of informable slot values and requestable slots. BLEU (Papineni et al., 2002) is applied to the generated agent responses for evaluating language quality. Although it is a poor choice for evaluating dialogue systems (Liu et al., 2016), we still report it in order to compare with previous work that has adopted it. For task completion evaluation, the Entity Match Rate (EMR) (Wen et al., 2017b) and Success F_1 score ($SuccF_1$) (Lei et al., 2018) are reported. EMR evaluates whether a system can correctly retrieve the user's indicated entity (record) from the KB based on the generated constraints so it can have only a score of 0 or 1 for each dialogue. The $SuccF_1$ score evaluates how a system responds to the user's requests at dialogue level: it is F_1 score of the response slots in the agent responses.

4.3 Benchmarks

We compare FSDM with four baseline methods and two ablations.

NDM (Wen et al., 2017b) proposes a modular end-to-end trainable network. It applies delexicalization on user utterances and responses.

LIDM (Wen et al., 2017a) improves over NDM by employing a discrete latent variable to learn underlying dialogue acts. This allows the system to be refined by reinforcement learning.

KVRN (Eric et al., 2017) adopts a copy-augmented Seq2Seq model for agent response generation and uses an attention mechanism on the KB.

It does not perform belief state tracking.

TSCP/RL (Lei et al., 2018) is a two-stage CopyNet which consists of one encoder and two copy-mechanism-augmented decoders for belief state and response generation. **TSCP** includes further parameter tuning with reinforcement learning to increase the appearance of response slots in the generated response. We were unable to replicate the reported results using the provided code[4], hyperparameters, and random seed, so we report both the results from the paper and the average of 5 runs on the code with different random seeds (marked with †).

FSDM is the proposed method and we report two ablations: in **FSDM/St** the whole state tracking is removed (informable, requestable and response slots) and the answer is generated from the encoding of the input, while in **FSDM/Res**, only the response slot decoder is removed.

4.4 Result Analysis

At the turn level, FSDM and FSDM/Res perform better than TSCP and TSCP/RL on belief state tracking, especially on requestable slots, as shown in Table 2. FSDM and FSDM/Res use independent binary classifiers for the requestable slots and are capable of predicting the correct slots in all those cases. FSDM/Res and TSCP/RL do not have any additional mechanism for generating response slot, so FSDM/Res performing better than TSCP/RL shows the effectiveness of flexible-structured belief state tracker. Moreover, FSDM performs better than FSDM/Res, but TSCP performs worse than TSCP/RL. This suggests that using RL to increase the appearance of response slots in the response decoder does not help belief state tracking, but our response slot decoder does.

FSDM performs better than all benchmarks on the dialogue level measures too, as shown in Table 3, with the exception of BLEU score on KVRET, where it is still competitive. Comparing TSCP/RL and FSDM/Res, the flexibly-structured belief state tracker achieves better task completion than the free-form belief state tracker. Furthermore, FSDM performing better than FSDM/Res shows the effectiveness of the response slot decoder for task completion. The most significant performance improvement is obtained on CamRest by FSDM, confirming that the additional inductive bias helps to generalize from smaller datasets. More importantly, the experiment confirms that, although making weaker assumptions that are reasonable for real-world applications, FSDM is capable of performing at least as well as models that make stronger limiting assumptions which make them unusable in real-world applications.

[4] https://github.com/WING-NUS/sequicity

4.5 Error Analysis

We investigated the errors that both TSCP and FSDM make and discovered that the sequential nature of the TSCP state tracker leads to the memorization of common patterns that FSDM is not subject to. As an example (Table 4), TSCP often generates "date; party" as requestable slots even if only "party" and "time" are requested like in "what time is my next activity and who will be attending?" or if "party", "time" and "date" are requested like in "what is the date and time of my next meeting and who will be attending it?". FSDM produces correct belief states in these examples.

FSDM misses some requestable slots in some conditions. For example, consider the user's utterance: "I would like their address and what part of town they are located in". The ground-truth requestable slots are 'address' and 'area'. FSDM only predicts 'address' and misses 'area', which suggests that the model did not recognize 'what part of town' as being a phrasing for requesting 'area'. Another example is when the agent proposes "the name_SLOT is moderately priced and in the area_SLOT part of town . would you like their location ?" and the user replies "i would like the location and the phone number, please". FSDM predicts 'phone' as a requestable slot, but misses 'address', suggesting it doesn't recognize the connection between 'location' and 'address'. The missing requestable slot issue may propagate to the agent response decoder. These issues may arise due to the use of fixed pre-trained embeddings and the single encoder. Using separate encoders for user utterance, agent response and dialogue history or fine-tuning the embeddings may solve the issue.

5 Conclusion

We propose the flexibly-structured dialogue model, a novel end-to-end architecture for task-oriented dialogue. It uses the structure in the schema of the KB to make architectural choices that introduce inductive bias and address the limitations of fully structured and free-form methods. The experiment suggests that this architecture is competitive

with state-of-the-art models, while at the same time providing a more practical solution for real-world applications.

Acknowledgments

We would like to thank Alexandros Papangelis, Janice Lam, Stefan Douglas Webb and SIGDIAL reviewers for their valuable comments.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations, San Diego, California, USA*.

Ankur Bapna, Gökhan Tür, Dilek Z. Hakkani-Tür, and Larry P. Heck. 2017. Towards zero-shot frame semantic parsing for domain scaling. In *INTERSPEECH*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media, Inc.

Antoine Bordes, Y-Lan Boureau, and Jason Weston. 2017. Learning end-to-end goal-oriented dialog. In *International Conference on Learning Representations, Toulon, France*.

Pawe Budzianowski, Iigo Casanueva, Bo-Hsiang Tseng, and Milica Gai. 2018. Towards end-to-end multi-domain dialogue modelling. *Technical Report CUED/F-INFENG/TR.706, Cambridge University*.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734. ACL.

Franck Dernoncourt, Ji Young Lee, Trung H. Bui, and Hung H. Bui. 2016. Robust dialog state tracking for large ontologies. In *IWSDS*, volume 427 of *Lecture Notes in Electrical Engineering*, pages 475–485. Springer.

Mihail Eric, Lakshmi Krishnan, Francois Charette, and Christopher D. Manning. 2017. Key-value retrieval networks for task-oriented dialogue. In *SIGDIAL Conference*, pages 37–49. Association for Computational Linguistics.

Pascale Fung, Chien-Sheng Wu, and Andrea Madotto. 2018. Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. In *ACL (1)*, pages 1468–1478. Association for Computational Linguistics.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *ACL (1)*. The Association for Computer Linguistics.

Matthew Henderson, Blaise Thomson, and Steve J. Young. 2014. Robust dialog state tracking using delexicalised recurrent neural networks and unsupervised adaptation. *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 360–365.

Takaaki Hori, Hai Wang, Chiori Hori, Shinji Watanabe, Bret Harsham, Jonathan Le Roux, John R. Hershey, Yusuke Koji, Yi Jing, Zhaocheng Zhu, and Takeyuki Aikawa. 2016. Dialog state tracking with attention-based sequence-to-sequence learning. In *SLT*, pages 552–558. IEEE.

Youngsoo Jang, Jiyeon Ham, Byung-Jun Lee, Youngjae Chang, and Kee-Eung Kim. 2016. Neural dialog state tracker for large ontologies by attention mechanism. *2016 IEEE Spoken Language Technology Workshop (SLT)*, pages 531–537.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, San Diego, California, USA*.

Yuka Kobayashi, Takami Yoshida, Kenji Iwata, Hiroshi Fujimura, and Masami Akamine. 2018. Out-of-domain slot value detection for spoken dialogue systems with context information. *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 854–861.

Wenqiang Lei, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He, and Dawei Yin. 2018. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In *ACL*.

Xiujun Li, Sarah Panda, Jingjing Liu, and Jianfeng Gao. 2018. Microsoft dialogue challenge: Building end-to-end task-completion dialogue systems. volume abs/1807.11125.

Bing Liu and Ian Lane. 2018. End-to-end learning of task-oriented dialogs. In *Proceedings of the NAACL-HLT*.

Bing Liu, Gokhan Tur, Dilek Hakkani-Tur, Pararth Shah, and Larry Heck. 2018. Dialogue learning with human teaching and feedback in end-to-end trainable task-oriented dialogue systems. In *NAACL*.

Chia-Wei Liu, Ryan Lowe, Iulian Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *EMNLP*, pages 2122–2132. The Association for Computational Linguistics.

Ryan Thomas Lowe, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. 2017. Training end-to-end dialogue systems with

the ubuntu dialogue corpus. *Dialogue and Discourse*, 8(1):31–65.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318. ACL.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543. ACL.

Osman Ramadan, Paweł Budzianowski, and Milica Gasic. 2018. Large-scale multi-domain belief tracking with knowledge sharing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 432–437.

Abhinav Rastogi, Dilek Hakkani-Tur, and Larry Heck. 2017. Scalable multi-domain dialogue state tracking. In *Proceedings of IEEE ASRU*.

Liliang Ren, Kaige Xie, Lu Chen, and Kai Yu. 2018. Towards universal dialogue state tracking. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2780–2786.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*, pages 3776–3784. AAAI Press.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *NIPS*, pages 2440–2448.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112.

Haoyang Wen, Yijia Liu, Wanxiang Che, Libo Qin, and Ting Liu. 2018. Sequence-to-sequence learning for task-oriented dialogue with dialogue state representation. pages 3781–3792.

Tsung-Hsien Wen, Yishu Miao, Phil Blunsom, and Steve J. Young. 2017a. Latent intention dialogue models. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3732–3741. PMLR.

Tsung-Hsien Wen, David Vandyke, Nikola Mrkšić, Milica Gasic, Lina M Rojas Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. 2017b. A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 438–449. ACL.

Jason D. Williams, Kavosh Asadi, and Geoffrey Zweig. 2017. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. In *ACL (1)*, pages 665–677. Association for Computational Linguistics.

Puyang Xu and Qi Hu. 2018a. An end-to-end approach for handling unknown slot values in dialogue state tracking. In *ACL (1)*, pages 1448–1457. Association for Computational Linguistics.

Puyang Xu and Qi Hu. 2018b. An end-to-end approach for handling unknown slot values in dialogue state tracking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1448–1457. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2018. Global-locally self-attentive encoder for dialogue state tracking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1458–1467. Association for Computational Linguistics.