Incorporating the Structure of the Belief State in End-to-End Task-Oriented Dialogue Systems

Lei Shu*, Piero Molino[2], Mahdi Namazifar[3], Bing Liu[1], Hu Xu[1], Huaixiu Zheng[3], and Gokhan Tur[3]

[1]University of Illinois at Chicago
[2]Uber AI Labs
[3]Uber Technologies Inc.
[1]*{lshu3, liub, hxu48}@uic.edu*
[2]*piero@uber.com*
[3]*{mahdin, huaixiu.zheng, gokhan}@uber.com*

Abstract

End-to-end trainable networks try to overcome error propagation, lack of generalization and overall brittleness of traditional modularized task-oriented dialogue-system architectures. Most proposed models expand on the sequence-to-sequence architecture. Some of them don't track belief state, which makes it difficult to interact with ever-changing knowledge bases, while the ones that explicitly track the belief state do it with classifiers. The use of classifiers suffers from the out-of-vocabulary words problem, making these models hard to use in real-world applications with ever-changing knowledge bases. We propose Structured Belief Copy Network (SBCN), a novel end-to-end trainable architecture that allows for interaction with external symbolic knowledge bases and solves the out-of-vocabulary problem at the same time. It explicitly uses the structure of the belief state in its architecture by using different sequential decoders with a copy mechanism for the different informable slots and a multi-label decoder for the requestable slots, and this provides better inductive bias. Through empirical results we show that SBCN achieves state-of-the-art results on dialogue datasets while providing a practical architecture that can be used in real-world applications.

1 Introduction

Traditional task-oriented dialogue systems take user's utterances as input and output the responses. Their architecture is a pipeline of functional modules: automatic speech recognition, natural language understanding, dialogue state tracking, dialogue policy engine, natural language generation and text-to-speech. Each module is learned separately and they are connected at prediction time by feeding the output of one module as input to the following one. This kind of architecture, although effective in dividing a complex problem into manageable sub-problems, suffers from error propagation throughout the pipeline. At the same time, it only partially accounts for uncertainty in the predictions of each module and requires considerable engineering effort, with cascading updates, when deployed in production. Finally, collecting supervision for training all the modules is expensive, so most dialogue datasets are relatively small in size, making it harder to train models that generalize well on unseen data.

To tackle these problems, recently end-to-end task-oriented dialogue systems have been proposed [13, 24, 14, 11, 12, 4, 3, 23, 19, among others]. Those systems train one whole text-in-text-out

*Work mostly performed as an intern at Uber AI Labs

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Table 1: Dialogue Example

user	Is there a cheap place in the north part of town?
agent	There are two restaurants that meet your specification. One is Italian and the other is Indian.
delexicalization	There are two restaurants that meet your specification. One is food_slot and the other is food_slot.
informable slot	price=cheap, area=north
requestable slot	
response slot	food_slot
user	Italian, please. Can you give me the address?
agent	Da Vinci pizzeria is in the north part of town their address is 20 Milton Road Chesterton.
delexicalization	name_slot is in the north part of town and their address is address_slot.
informable Slot	price=cheap, area=north, food=italian
requestable slot	address
response slot	name_slot, address_slot
user	What is the phone number please?
agent	da vinci pizzeria's phone number is 01223351707.
delexicalization	name_slot 's phone number is phone_slot.
informable slot	price=cheap, area=north, food=italian
requestable slot	phone
response slot	name_slot, phone_slot

model which reads user's utterances and generates responses. The Sequence-to-sequence (Seq2Seq) architecture [21] is also utilized as it fits the problem formulation and has been shown to have competitive performance on the language understanding and generation. However, vanilla Seq2Seq lacks the ability of incorporating external knowledge sources and managing dialogue history, which makes it more suitable for chit-chat dialogue than for task-oriented dialogue.

In order to adapt Seq2Seq to task-oriented dialogue and mitigate its shortcomings, several architectures have been proposed [23, 13, 12, 11, 6, 7] that are end-to-end trainable while keeping all functional modules. In most dialogue systems with belief state tracking, the belief state is defined in terms of informable slot values, the values the user informs the system about (e.g. $informable = \{price = cheap, food = italian\}$ if they want to book a cheap Italian restaurant), and requestable slots, the slots the user wants to know about (e.g. $requestable = \{address, phone\}$ if they want to know the address and the phone number of the restaurant they are booking). In particular, in [9] the authors add belief state tracking to the Seq2Seq architecture. Their model represents belief states as sequences of informable slot values and requestable slots names and encodes and decodes them with CopyNets [8]. This approach has the advantage of solving the out-of-vocabulary words problem, but the lack of structure in the way belief states are encoded and decoded both introduces unwanted dependencies between slot values and makes it hard to identify informable slot value assignments, making it impossible, for instance, to have two slots with overlapping value sets.

We propose Structured Belief Copy Network (SBCN) as a way to introduce a more structured architecture for performing belief state decoding that provides better inductive bias, thus making training easier. In our proposal, each slot has its own decoder that generates or copies the value of informable slots, as this allows for both out-of-vocabulary values and multi-word expressions like in [9], but also univocally identifies the value of each slot. As for requestable slots, the decoder is a multi-label classifier that performs a binary classification for each slot. As the slots that appear in the response may be different from the ones the user requested, we add a response slot binary classifier that predicts the chances of each slot appear in the response. This architecture achieves state-of-the-art results on the Cambridge Restaurant dataset [23] without the need for fine tuning through reinforcement learning, suggesting that injecting this form of structured architectural bias in the model strikes a better balance between flexibility and structure.

2 Related Work

End-to-end task-oriented dialogue systems is a novel yet vibrant research area [13, 24, 14, 11, 12, 4, 3, among others]. Our work is closely related to task-oriented dialogue systems that extend the Seq2Seq architecture [6, 7]. In these works, the architecture contains a sequential encoder for user utterances and a sequential decoder for response generation, but no belief tracking is explicitly performed. Conversely, in this work we explicitly include belief tracking inside the model architecture. On the other hand, these works incorporate external knowledge bases by encoding them in key-value pairs of learned representations and then query these pairs using an attention mechanism [1, 15]. In

particular, in [7] the authors adopt Memory Networks [20] to memorize the retrieved knowledge base entities and words appearing in dialogue history. This means that the training and inference of the model scales at least linearly with the size of the knowledge base and implies that the model needs to be retrained at each update of the knowledge base, both of which are issues that make these approaches less practical in real-world applications. In [6] the authors adopt a copy mechanism that allows copying retrieved record information to the generated response, which alleviates Seq2Seq's problem of having to generate the lexicalized requested slot values of the retrieved entities. The fact that our approach explicitly tracks the belief state makes it possible for us to query the knowledge base symbolically and to perform de-lexicalized response generation, as we substitute de-lexicalized slot names in the generated response with the values in the knowledge base query result.

Our work is also akin to modularly connected end-to-end trainable networks [23, 22, 13, 12, 11]. [23] includes belief state tracking, but has two phases in training: the first phase uses belief state supervision and then the second phase uses response generation supervision. [22] adds a policy network using latent representations to [23] so that the dialogue system can be continuously improved through reinforcement learning after the model is optimized with the original training data. These methods utilize classification as a way to decode the belief state, which makes them suffer from the out-of-vocabulary words problem, as users may mention values for the informable slots which have never appeared in training dataset. Furthermore, it cannot deal with growing knowledge bases, as if a new value is added to any of the slots a new class has to be added to the belief state classifier. Our choice of generating and copying values of the informable slots through a CopyNet [8] addresses these issues. Another main difference in our work is the presence of a multi-label classifier for the response slot that predicts the probability of each slot appearing in the final response. The main advantage of this classifier is that the slots present in the de-lexicalized response are not always the only ones requested by the user, so directly predicting which ones will appear better informs the copy mechanism of the response decoder.

Finally, in [9] the authors adopt a CopyNet for decoding the belief state as well as the response. This approach has the advantage of solving several problems such as the presence of out-of-vocabulary words in slots values and deals naturally with multi-word phrases. On the other hand, the architecture has also some downsides. Keeping track of the informable slot values without an explicit assignment to a specific informable slot makes it difficult to use this approach when different slots may have overlapping sets of values, e.g. departure and destination airport in a travel booking system. Moreover, the fact that the order of the sequence in which both informable slot values and requestable slots are encoded and decoded is arbitrary suggests that the sequential architecture adopted may not give the right inductive bias. Our proposed method solves those problems by introducing structure in the architecture used to decode the belief state, using slot-specific CopyNets for informable slots and multi-label classification for requestable slots.

3 Methodology

We name our model Structured Belief Copy Network (SBCN) as we introduce a structured architectural bias in the belief state tracking. Our basic assumption is that dialogues have the markov property: current-turn decisions (predicted new belief state and generated agent response) are independent of history given the previous-turn agent response and belief state. The previous-turn agent response and belief state contains all the information about the past needed to generate an answer. The exposition will use a prototypical Cambridge Restaurant dataset dialogue turn as a running example to contextualize our architectural decisions.

The overall architecture contains five components as shown in Figure 1: an input encoder, a belief state tracker (informable slot values decoder and requestable slot binary classifier), a knowledge base query component, a response slot binary classifier and a response decoder. The input encoder encodes the concatenation of the previous-turn tokenized agent response A_{t-1}, the previous-turn belief state B_{t-1} and the current-turn tokenized user utterance U_t. The belief state is provided as a sequence of informable slot name tokens with their respective value and requestable slot names that are requested by the user. For instance the state $\{price = cheap, area = north, food = italian; address = 1, phone = 1, postcode = 0\}$ is encoded as the sequence $\langle cheap, end_{price}, north, end_{area}, italian, end_{food}, address, phone, end_{belief}\rangle$ where end_* is a token that denotes the end of the sequence contaning the value of an informable slot and end_{belief} denotes the end of the seuqnce representing the belief state. The last hidden state of the encoder serves

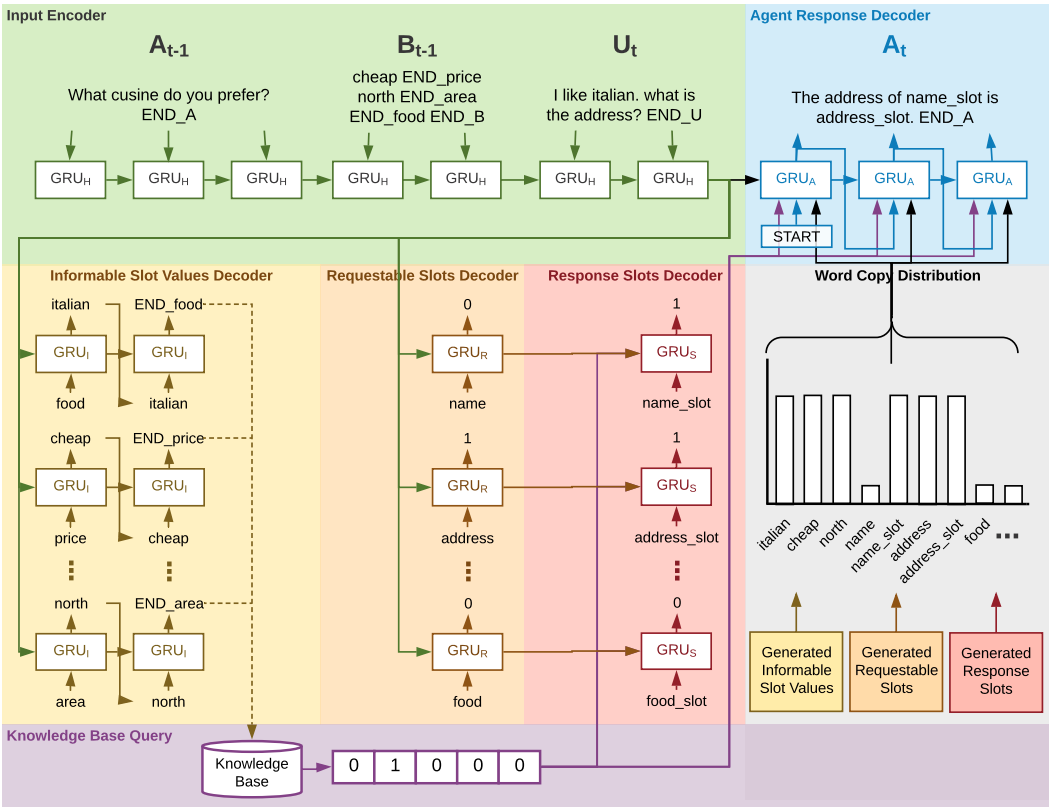

Figure 1: SBCN architecture contains an input encoder (green), a belief state tracker (yellow for the informable slot values, orange for the requestable slots), a knowledge base query component (purple), a response slot classifier (red), a component that calculates word copy probability (grey) and a response decoder (blue). Attention connections are not drawn for the sake of clarity.

as the initial hidden state of the belief state tracker and the response decoder. The belief state tracker contains two modules: the informable slot value decoder and the requestable slot binary classifier. The informable slot value decoder generates the constraints I_t that are used for performing a query to the knowledge base, while the requestable slot binary classifier identifies the slots that the user requested R_t. The belief state of the current-turn t is $B_t = \{I_t, R_t\}$. Given the constraints obtained from the generated informable slot values I_t, the knowledge base query component performs a query on the knowledge base and encodes the number of records returned in a one-hot vector d_t. The response slot binary classifier predicts which slots should appear in the agent response S_t. Finally, the agent response decoder takes in the knowledge base output d_t, a word copy probability vector $\mathcal{P}^c$ computed from I_t, R_t, S_t together with an attention of the input encoder hidden states and the belief decoders hidden states, for generating a response A_t. Both requestable slot and response slot binary classifiers share weights across slots, making both them logically equivalent to multi-label classifiers. In the rest of this section each component will be described in detail.

3.1 Input Encoder

The input encoder consists of an embedding layer followed by a recurrent layer with Gated Recurrent Units (GRU) [5]. 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. Let h_l^E denote the last hidden state of the encoder.

3.2 Informable Slot Value Decoder

The informable slot values are values provided by the user that constitute constraints used to search for relevant entities in the knowledge base. For instance, the user utterance "I'd like a cheap restaurant" informs that the agent has to search in the knowledge base for all entities where $\{price = cheap\}$.

The informable slot value decoder is modeled as a multiple-head structure consisting of GRU recurrent layers combined with a copy mechanism [8]. They are the cells shown in the yellow section of Figure 1. The **multiple-head structure** is composed of tied-weights GRU generators that take the same initial hidden state h_l^E, but have different start-of-sentence symbols for each different informable slot value. This way, each informable slot value decoder is dependent on the encoder's output, but it is independent from the values generated for the other slots. For example, the value "italian" of the food slot is independent from the generate price range "cheap", while in above mentioned literature the generated values of informable slots are often sequentially dependent. Let $\{k^I\}$ denote the set of informable slots. The copy mechanism [8] helps the decoder in generating words that appeared in the input encoder. Words can be copied from $A_{t-1} \circ B_{t-1} \circ U_t$. The probability of the j^{th} word $P(y_j^{k^I})$ is calculated as follows: (1) calculate the copy score $\phi_c(y_j^{k^I})$ and generation score $\phi_g(y_j^{k^I})$ based on the hidden state $h_j^{k^I}$, (2) sum the two scores and obtain probabilities through a softmax:

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

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. Note that we follow [8] and [1] for our copy mechanism and attention mechanism implementation respectively.

Let z denote the ground truth label. The loss for the informable slot values decoder is calculated as as follows where Y^{K^I} is the sequence of informable slot value decoder predictions:

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

3.3 Requestable Slot Binary Classifier

The requestable slots are the attributes of knowledge base entries that are explicitly requested by the user. For instance the user utterance "What's the restaurant address and telephone number?" is requesting for $\{address, telephone\}$ slots of a restaurant.

A binary classifier is used for deciding whether the user requested each slot or not, performing a multi-label classification. Let $\{k^R\}$ denote the set of requestable slots. A one-step GRU is used to perform the classification. Given the initial state h_l^E and the embedding vector e^{k^R} of one requestable slot k^R, it is used to pay attention to the input encoder hidden vectors to compute a context vector c^{k^R}. We use the attention mechanism described in [16]. The concatenation of c^{k^R} and e^{k^R} is passed as input and h_l^E as 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 a probability y^{k^R}, 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:

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

3.4 Knowledge Base Query

The generated informable slot values $I_t = \{Y^{k^I}\}$ are used as constraints of the symbolic knowledge base query. The knowledge base is composed of one or more relational tables and each entity is a tuple in one table. In the restaurant booking domain there is only one table containing information about restaurant. 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, one-hot vector d_t, indicates the number of records in the knowledge base that satisfy 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 results and it is later used to inform the response slot binary classifier and the agent response decoder.

3.5 Response Slot Binary Classifier

The response slots are the slots that are actually used in the de-lexicalized agent response. When the response is presented to the user, they are replaced by actual values of the database entries. They are dependent on the slots that were requested in the user utterance as well as the results obtained from the knowledge base query. For instance, the user utterance "What's the address?" will map to the requestable slot "address", while the agent may generate the response "name_slot is located in address_slot in the area_slot part of town" An example of how the results obtained from querying the knowledge base can impact the response slots is when multiple entities are returned: the response may contains several "name_slot" and "food_slot".

The response slot binary classifier is similar to the requestable slot binary classifier. The response slots $\{k^S\}$ map one-to-one to the requestable slots $\{k^R\}$, e.g. "address_slot" maps to "address". The initial state of each response slot decoder is the last hidden state of the requestable slot decoder it maps to. 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 knowledge base query vector d_t are used as input to a one-step GRU. 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 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:

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

3.6 Word Copy Probability and Agent Response Decoder

To allow the copy mechanism of the agent response decoder to consider the outputs of the informable slot values, requestable slots and response slots decoders, a vector of independent word copy probabilities $\mathcal{P}^C$ 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 agent response decoder prediction probability to generate the response.

The agent response decoder is responsible for generating the final de-lexicalized agent response to be returned to the user. To this end, the response slots are substituted with the values of the results obtained by querying the knowledge base before the response is returned. Note that this component jointly models the policy engine and natural language generation in a conventional modular dialogue system [23]. The actions taken by the policy engine are hence latent depending on the belief state and KB output. This is in contrast to chit-chat systems where there is no action. However, similar to modular systems, the responses generated according to policy actions are usually either information bearing or requesting for more information from 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:

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

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 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 previous generated word and the knowledge base 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 the softmax of the sum of generation score and copy score:

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

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:

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

As the agent response is de-lexicalized, it will contain slot names. They will be resolved against the results returned by the knowledge base. For instance if the generated answer is "The address of name_slot is address_slot" and the returned tuple from the query to the knowledge base is $\{name = DaVinci, address = 123BakerStreet, area = north, ...\}$, the final resolved answer will be: "The address of DaVinci is 123 Baker Street".

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 answer 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 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 in order to compute the d_t, while at prediction time results obtained by using the generated informable slot values I_t are used.

4 Experiment

4.1 Setup

We tested the SBCN on Cambridge Restaurant dataset (CamRest) [23] which is composed of 408, 136 and 136 dialogues for training, validation and testing respectively and contains 99 unique records in the knowledge base.

We use NLTK [2] to tokenize each sentence. The user utterances are exactly the original texts, while all agent response texts are de-lexicalized as described in [9]. In the CamRest dataset, annotations about the informable slot values and requestable slots for each turn are provided. We obtain the labels for the response slot decoder from the de-lexicalized response texts. For embedding words, 300-dimensional GloVe embeddings [18] trained on 840B words from Common Crawl are used. For tokens not present in the GloVe set, they are initialized to be the average of all other embeddings plus a small amount of random noise to make them different from each other, and they are updated during the training.

We evaluate the performance in terms of 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 [17] is applied on the generated agent responses for evaluating language quality. For task completion evaluation, Entity Match Rate (EMR) [23] and Success F_1 score (SuccF_1) [9] are reported. EMR evaluates whether a system can correctly retrieve the user's indicated entity (record) from the knowledge base 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 the F_1 score of the response slots in the agent responses.

We optimize both training and model hyperparameters by running bayesian optimization over the validation set EMR using skopt[2]. The model that performed the best on the validation set uses the Adam optimizer [10] with a learning rate of 0.00025 for minimizing the loss in Equation 12. We apply dropout with a rate of 0.45 after the embedding layer, the GRU layer and any linear layer. The dimension of all hidden states is 128. Loss weights α^I, α^R, α^S, α^A are 1.5, 9.5, 8.5, 0.3 respectively.

We compare SBCN with the following four methods.

[2] `https://scikit-optimize.github.io/`

Method	Inf P	Inf R	Inf F_1	Req P	Req R	Req F_1	BLEU	EMR	SuccF_1
NDM	**0.998**	0.961	0.979	0.987	0.938	0.962	0.212	0.904	0.832
LIDM	-	-	-	-	-	-	0.246	0.912	0.840
KVRN	-	-	-	-	-	-	0.134	-	-
TSCP	-	-	-	-	-	-	0.253	0.927	**0.854**
TSCP†	0.970	0.971	0.971	0.983	0.935	0.959	0.237	0.915	0.826
TSCP+RL†	0.970	0.971	0.971	0.983	0.938	0.960	0.237	0.913	0.841
SBCN	0.982*	**0.984***	**0.983***	**0.996***	**0.952**	**0.974***	**0.254***	**0.933***	0.852*

Table 2: Comparison of SBCN perfromance with the baselines on the CamRest dataset. **Inf**: Informable, **Req**: Requestable, **P**: Precision, **R**: Recall, **SuccF_1**: Success F_1 score, **EMR**: Entity Match Rate. Results marked with † are computed using available code, all the other ones are reported from the original papers. * indicates the result is statistically significant at the level of 0.05 comparing to TSCP+RL†.

NDM [23] proposes a modularly-connected end-to-end trainable network. It is composed of an intent network, a belief tracker, a knowledge base operator, a policy network and a response generation module. It applies de-lexicalization on both user utterance and agent response.

LIDM [22] 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 [6] adopts a copy-augmented Seq2Seq model for agent response generation and uses an attention mechanism on the knowledge base. It does not perform belief state tracking.

TSCP [9] propose a two stage CopyNet which consists of one encoder and two copy-mechanism [8] augmented decoders. The first decoder decodes the belief state, including informable slot values and requestable slots. The second decoder generates the response based on the first decoder's generated belief state and the results of a knowledge base query. **TSCP+RL** includes further parameter tuning with reinforcement learning. We were unable to replicate the results reported in the paper using the provided code[3], hyperparameter setting and random seed, so we report both the results from the paper and the best of 5 runs on the code given different random seeds (marked with †).

4.2 Result Analysis

As the results in Table 2 show, SBCN performs better than the benchmarks on all the measures except SuccF_1, where it is still competitive. In particular, SBCN performs better than LIDM and NDM on the belief tracking measures (Inf. and Req.): it is particularly interesting because those models delexicalize both the user utterance and the agent response, which substantially helps the performance on the informable slot, but assumes that perfect delexicalization is performed as a preprocessing step, while SBCN does not require this additional step, making it more practical. SBCN also performs better than both TSCP† and TSCP+RL†: the BLEU score is slightly better than the best TSCP† one and the SuccF_1 is slightly better than the best TSCP+RL†, but the biggest advantage is in the EMR. This is not surprising given that the EMR is the metric we optimized the hyperaprameters for. Finally, compared to the reported TSCP scores, both BLEU and SuccF_1 are within 0.002, a negligible difference, while SBCN outperforms TSCP on EMR.

5 Conclusion

In this paper, we propose Structured Belief Copy Network, a novel architecture for task-oriented end-to-end dialogue. It explicitly uses the structure of the belief state for guiding the design of an architecture that provides better inductive bias and addresses the limitations of previous works. In particular, it naturally deals with the out-of-vocabulary problem by adopting a copy mechanism and by predicting the belief state making it possible to query ever-changing knowledge bases, both characteristics needed to make the use of the model practical in real world applications. The experiment on the Cambridge Restaurant dataset suggests that this architecture is also competitive with state-of-the-art models.

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

References

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

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

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

[4] Paweł Budzianowski, Iñigo Casanueva, Bo-Hsiang Tseng, and Milica Gašić. Towards end-to-end multi-domain dialogue modelling. *Technical Report CUED/F-INFENG/TR.706, Cambridge University*, 2018.

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

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

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

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

[9] Min-Yen Kan, Xiangnan He, Wenqiang Lei, Xisen Jin, Zhaochun Ren, and Dawei Yin. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In *ACL (1)*, pages 1437–1447. Association for Computational Linguistics, 2018.

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

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

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

[13] Bng Liu and Ian Lane. End-to-end learning of task-oriented dialogs. In *Proceedings of the NAACL-HLT*, 2018.

[14] Ryan Thomas Lowe, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue and Discourse*, 8(1):31–65, 2017.

[15] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, pages 1412–1421. The Association for Computational Linguistics, 2015.

[16] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, pages 1412–1421. The Association for Computational Linguistics, 2015.

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

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

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

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

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

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

[23] Tsung-Hsien Wen, David Vandyke, Nikola Mrkšić, Milica Gasic, Lina M Rojas Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. 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, 2017.

[24] Jason D. Williams, Kavosh Asadi, and Geoffrey Zweig. 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, 2017.