Sequicity: Simplifying Task-oriented Dialogue Systems with Single Sequence-to-Sequence Architectures

Wenqiang Lei[‡*], **Xisen Jin**[§*], **Zhaochun Ren**[†], **Xiangnan He**[‡], **Min-Yen Kan**[‡], **Dawei Yin**[†]
‡National University of Singapore, Singapore
§Fudan University, Shanghai, China
†Data Science Lab, JD.com, Beijing, China
{wenqianglei,xisenjin}@gmail.com renzhaochun@jd.com
kanmy@comp.nus.edu.sg xiangnanhe@gmail.com yindawei@acm.org

Abstract

Existing solutions to task-oriented dialogue systems follow pipeline designs which introduce architectural complexity and fragility. We propose a novel, holistic, extendable framework based on a single sequence-to-sequence (seq2seq) model which can be optimized with supervised or reinforcement learning. A key contribution is that we design text spans named *belief spans* to track dialogue believes, allowing task-oriented dialogue systems to be modeled in a seq2seq way. Based on this, we propose a simplistic *Two Stage CopyNet* instantiation which demonstrates good scalability: significantly reducing model complexity in terms of number of parameters and training time by an order of magnitude. It significantly outperforms state-of-the-art pipeline-based methods on two datasets and retains a satisfactory entity match rate on out-of-vocabulary (OOV) cases where pipeline-designed competitors totally fail.

1 Introduction

The challenge of achieving both task completion and human-like response generation for task-oriented dialogue systems is gaining research interest. Wen et al. (2017b, 2016a, 2017a) pioneered a set of models to address this challenge. Their proposed architectures follow traditional pipeline designs, where the belief tracking component is the key component (Chen et al., 2017).

In the current paradigm, such a belief tracker builds a complex multi-class classifier for each slot (See §3.2) which can suffer from high complexity, especially when the number of slots and their values grow. Since all the possible slot values have to be pre-defined as classification labels, such trackers also cannot handle the requests that have out-of-vocabulary (OOV) slot values. Moreover, the belief tracker requires delexicalization, i.e., replacing slot values with their slot names in utterances (Mrkšić et al., 2017). It does not scale well, due to the lexical diversity. The belief tracker also needs to be pre-trained, making the models unrealistic for end-to-end training (Eric and Manning, 2017a). While Eric and Manning (2017a,b) investigated building task-oriented dialogue systems by using a seq2seq model, unfortunately, their methods are rather preliminary and do not perform well in either task completion or response generation, due to their omission of a belief tracker.

Questioning the basic pipeline architecture, in this paper, we re-examine the tenets of belief tracking in light of advances in deep learning. We introduce the concept of a *belief span* (bspan), a text span that tracks the belief states at each turn. This leads to a new framework, named *Sequicity*, with a single seq2seq model. Sequicity decomposes the task-oriented dialogue problem into the generation of bspans and machine responses, converting this problem into a sequence optimization problem. In practice, Sequicity decodes in two stages: in the first stage, it decodes a bspan to facilitate knowledge base (KB) search; in the second, it decodes a machine response on the condition of knowledge base search result and the bspan.

Our method represents a shift in perspective compared to existing work. Sequicity employs a single seq2seq model, resulting in a vastly simplified architecture. Unlike previous approaches with an overly parameterized delexicalization-based belief tracker, Sequicity achieves much less train-

* Work performed during an internship at Data Science Lab, JD.com.

ing time, better performance on larger a dataset and an exceptional ability to handle OOV cases. Furthermore, Sequicity is a theoretically and aesthetically appealing framework, as it achieves true end-to-end trainability using only one seq2seq model. As such, Sequicity leverages the rapid development of seq2seq models (Gehring et al., 2017; Vaswani et al., 2017; Yu et al., 2017) in developing solutions to task-oriented dialogue scenarios. In our implementation, we improve on CopyNet (Gu et al., 2016) to instantiate Sequicity framework in this paper, as key words present in bspans and machine responses recur from previous utterances. Extensive experiments conducted on two benchmark datasets verify the effectiveness of our proposed method.

Our contributions are fourfold: (1) We propose the *Sequicity* framework, which handles both task completion and response generation in a single seq2seq model; (2) We present an implementation of the Sequicity framework, called *Two Stage CopyNet* (*TSCP*), which has fewer number of parameters and trains faster than state-of-the-art baselines (Wen et al., 2017b, 2016a, 2017a); (3) We demonstrate that TSCP significantly outperforms state-of-the-art baselines on two large-scale datasets, inclusive of scenarios involving OOV; (4) We release source code of TSCP to assist the community to explore Sequicity[1].

2 Related Work

Historically, task-oriented dialog systems have been built as pipelines of separately trained modules. A typical pipeline design contains four components: 1) a user intent classifier, 2) a belief tracker, 3) a dialogue policy maker and a 4) response generator. User intent detectors classify user utterances to into one of the pre-defined intents. SVM, CNN and RNN models (Silva et al., 2011; Hashemi et al., 2016; Shi et al., 2016) perform well for intent classification. Belief trackers, which keep track of user goals and constraints every turn (Henderson et al., 2014a,b; Kim et al., 2017) are the most important component for task accomplishment. They model the probability distribution of values over each slot (Lee, 2013). Dialogue policy makers then generate the next available system action. Recent experiments suggest that reinforcement learning is a promising paradigm to accomplish this task (Young et al., 2013a; Cuayáhuitl et al., 2015; Liu and Lane, 2017), when state and action spaces are carefully designed (Young et al., 2010). Finally, in the response generation stage, pipeline designs usually pre-define fixed templates where placeholders are filled with slot values at runtime (Dhingra et al., 2017; Williams et al., 2017; Henderson et al., 2014b,a). However, this causes rather static responses that could lower user satisfaction. Generating a fluent, human-like response is considered a separate topic, typified by the topic of conversation systems (Li et al., 2015).

[1] http://github.com/WING-NUS/sequicity

3 Preliminaries

3.1 Encoder-Decoder Seq2seq Models

Current seq2seq models adopt encoder–decoder structures. Given a source sequence of tokens $X = x_1 x_2 ... x_n$, an encoder network represent X as hidden states: $\mathbf{H}^{(x)} = \mathbf{h}_1^{(x)} \mathbf{h}_2^{(x)} ... \mathbf{h}_n^{(x)}$. Based on $\mathbf{H}^{(x)}$, a decoder network generates a target sequence of tokens $Y = y_1 y_2 ... y_m$ whose likelihood should be maximized given the training corpus.

As of late, the recurrent neural network with attention (Att-RNN) is now considered a baseline encoder–decoder architecture. Such networks employ two (sometimes identical) RNNs, one for encoding (i.e., generating $\mathbf{H}^{(x)}$) and another for decoding. Particularly, for decoding y_j, the decoder RNN takes the embedding $\mathbf{y}_{j-1}$ to generate a hidden vector $\mathbf{h}_j^{(y)}$. Afterwards, the decoder attends to X: calculating attention scores between all $\mathbf{h}_i^{(x)} \in \mathbf{H}^{(x)}$ and $\mathbf{h}_j^{(y)}$ (Eq. (1)), and then sums all $\mathbf{h}_i^{(x)}$, weighted by their corresponding attention scores (Eqs. (2)). The summed result $\tilde{\mathbf{h}}_j^{(x)}$ concatenates $\mathbf{h}_j^{(y)}$ as a single vector which is mapped into an output space for a $softmax$ operation (Eq. (3)) to decode the current token:

$$u_{ij} = \mathbf{v}^T tanh(\mathbf{W}_1 \mathbf{h}_i^{(x)} + \mathbf{W}_2 \mathbf{h}_j^{(y)}) \quad (1)$$

$$\tilde{\mathbf{h}}_j^{(x)} = \sum_{i=1}^{n} \frac{e^{u_{ij}}}{\sum_i e^{u_{ij}}} \mathbf{h}_i^{(x)} \quad (2)$$

$$y_j = softmax(\mathbf{O} \begin{bmatrix} \tilde{\mathbf{h}}_j^{(x)} \\ \mathbf{h}_j^{(y)} \end{bmatrix}) \quad (3)$$

where $\mathbf{v} \in \mathbb{R}^{1 \times l}$; $\mathbf{W}_1$, $\mathbf{W}_2 \in \mathbb{R}^{l \times d}$ and $\mathbf{O} \in \mathbb{R}^{|V| \times d}$. d is embedding size and V is vocabulary set and $|V|$ is its size.

3.2 Belief Trackers

In the multi-turn scenarios, a belief tracker is the key component for task completion as it records key information from past turns (Wen et al., 2017b; Henderson et al., 2013, 2014a,b). Early belief trackers are designed as Bayesian networks where each node is a dialogue belief state (Paek and Horvitz, 2000; Young et al., 2013b). Recent work successfully represents belief trackers as discriminative classifiers (Henderson et al., 2013; Williams, 2012; Wen et al., 2017b).

Wen et al. (2017b) apply discrimination approaches (Henderson et al., 2013) to build one classifier for each slot in their belief tracker. Following the terminology of (Wen et al., 2017b), a slot can be either **informable** or **requestable**, which have been annotated in CamRes676 and KVRET. Individually, an informable slot, specified by user utterances in previous turns, is set to a constraint for knowledge base search; whereas a requestable slot records the user's need in the current dialogue. As an example of belief trackers in CamRes676, `food_type` is an informable slot, and a set of food types is also predefined (e.g., `Italian`) as corresponding slot values. In (Wen et al., 2017b), the informable slot `food_type` is recognized by a classifier, which takes user utterances as input to predict if and which type of food should be activated, while the requestable slot of `address` is a binary variable. `address` will be set to true if the slot is requested by the user.

4 Method

We now describe the Sequicity framework, by first explaining the core concept of bspans. We then instantiate the Sequicity framework with our introduction of an improved CopyNet (Gu et al., 2016).

4.1 Belief Spans for Belief Tracking

The core of belief tracking is keeping track of informable and requestable slot values when a dialogue progresses. In the era of pipeline-based methods, supervised classification is a straightforward solution. However, we observe that this traditional architecture can be updated by applying seq2seq models directly to the problem. In contrast to (Wen et al., 2017b) which treats slot values as classification labels, we record them in a text span, to be decoded by the model. This leverages the state-of-the-art neural seq2seq models to learn and dynamically generate them. Specifically, our bspan has an information field (marked with `<Inf></Inf>`) to store values of informable slots since only values are important for knowledge base search. Bspans can also feature a requested field (marked with `<Req></Req>`), storing requestable slot names if the corresponding value is `True`.

At turn t, given the user utterance U_t, we show an example of both bspan B_t and machine response R_t generation in Figure 1, where annotated slot values at each turn are decoded into bspans. B_1 contains an information slot `Italian` because the user stated "Italian food" in U_1. During the second turn, the user adds an additional constraint `cheap` resulting in two slot values in B_2's information field. In the third turn, the user further asks for the restaurant's phone and address, which are stored in requested slots of B_3.

Our bspan solution is concise: it simplifies multiple sophisticated classifiers with a single sequence model. Furthermore, it can be viewed as an explicit data structure that expedite knowledge base search as its format is fixed: following (Wen et al., 2017b), we use the informable slots values directly for matching fields of entries in databases.

4.2 The Sequicity Framework

We make a key observation that at turn t, a system only needs to refer to B_{t-1}, R_{t-1} and U_t to generate a new belief span B_t and machine response R_t, without appealing to knowing all past utterances. Such Markov assumption allows Sequicity to concatenate B_{t-1}, R_{t-1} and U_t (denoted as $B_{t-1}R_{t-1}U_t$) as a source sequence for seq2seq modeling, to generate B_t and R_t as target output sequences at each turn. More formally, we represent the dialogue utterances as $\{(B_0R_0U_1; B_1R_1); (B_1R_1U_2; B_2R_2); ...; (B_{t-1}R_{t-1}U_t; B_tR_t)\}$ where B_0 and R_0 are initialized as empty sequences. In this way, Sequicity fulfills both task accomplishment and response generation in an unified seq2seq model. Note that we process B_t and R_t separately, as the belief state B_t depends only on $B_{t-1}R_{t-1}U_t$, while the response R_t is additionally conditioned on B_t and the knowledge base search results (denoted as $\mathbf{k}_t$); that is, B_t informs the R_t's contents. For example, R_t must include all the request slots from B_t when communicating the entities fulfilling the requests found in the knowledge base. Here, $\mathbf{k}_t$ helps generate R_t pragmatically.

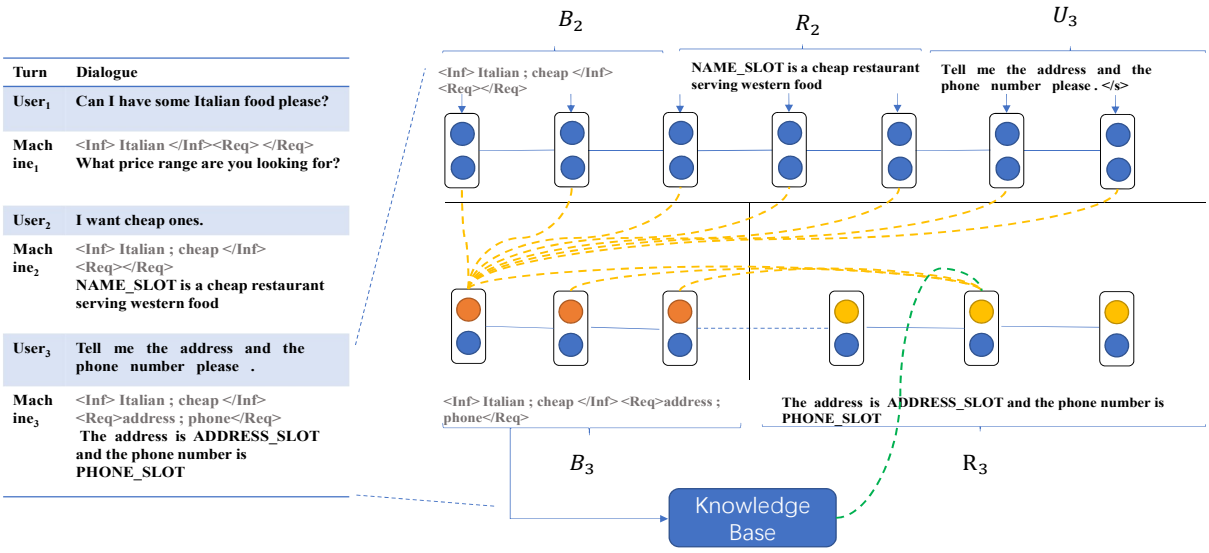


Figure 1: Sequicity overview. The left shows a sample dialogue; the right illustrates the Sequicity. B_t is employed only by the model, and not visible to users. During training, we substitute slot values with placeholders bearing the slot names for machine response. During testing, this is inverted: the placeholders are replaced by actual slot values, according to the item selected from the knowledge base.

Generally, $\mathbf{k}_t$ has three possibilities: 1) multiple matches, 2) exact match and 3) no match, while the machine responses differ accordingly. As an example, let's say a user requests an Italian restaurant. In the scenario of multiple matches, the system should prompt for additional constraints for disambiguation (such as restaurant price range). In the second exact match scenario where a single target (i.e., restaurant) has been found, the system should inform the user their requested information (e.g., restaurant address). If no entity is obtained, the system should inform the user and perhaps generate a cooperative response to retry a different constraint.

We thus formalize Sequicity as a seq2seq model which encodes $B_{t-1}R_{t-1}U_t$ jointly, but decodes B_t and R_t separately, in two serial stages. In the first stage, the seq2seq model decodes B_t unconditionally (Eq. 4a). Once B_t obtained, the decoding pauses to perform the requisite knowledge base search based on B_t, resulting in $\mathbf{k}_t$. Afterwards, the seq2seq model continues to the second decoding stage, where R_t is generated on the additional conditions of B_t and $\mathbf{k}_t$ (Eq. 4b).

$$B_t = \mathbf{seq2seq}(B_{t-1}R_{t-1}U_t|0,0) \quad (4a)$$

$$R_t = \mathbf{seq2seq}(B_{t-1}R_{t-1}U_t|B_t,\mathbf{k}_t) \quad (4b)$$

Sequicity is a general framework suitably implemented by any of the various seq2seq models. The additional modeling effort beyond a general seq2seq model is to add the conditioning on B_t and $\mathbf{k}_t$ to decode the machine response R_t. Fortunately, natural language generation with specific conditions has been extensively studied (Wen et al., 2016b; Karpathy and Fei-Fei, 2015; Mei et al., 2016) which can be employed within this framework.

4.3 Sequicity Instantiation: A Two Stage CopyNet

Although there are many possible instantiations, in this work we purposefully choose a simplistic architecture, leaving more sophisticated modeling for future work. We term our instantiated model a *Two Stage CopyNet* (*TSCP*). We denote the first m' tokens of target sequence Y are B_t and the rests are R_t, i.e. $B_t = y_1...y_{m'}$ and $R_t = y_{m'+1}...y_m$.

Two-Stage CopyNet. We choose to improve upon CopyNet (Gu et al., 2016) as our seq2seq model. This is a natural choice as we observe that target sequence generation often requires the copying of tokens from the input sequence. Let's discuss this in more detail. From a probabilistic point of view, the traditional encoder–decoder structure learns a language model. To decode y_j, we can employ a $softmax$ (e.g., Eq. 3) to calculate the probability distribution over V i.e., $P_j^g(v)$ where $v \in V$, and then choose the token with the highest generation probability. However, in our case, tokens in the target sequence Y might

be exactly copied from the input X (e.g., "Italian"). These copied words need to be explicitly modeled. CopyNet (Gu et al., 2016) is a natural fit here, as it enlarges the decoding output space from V to $V \cup X$. For y_j, it considers an additional copy probability $P_j^c(v)$, indicating the likelihood of y_j copied from $v \in X$. Following (Gu et al., 2016), the simple summation of both probabilities $P_j(v) = P_j^g(v) + P_j^c(v), v \in V \cup X$ is treated as the final probability in the original paper.

In Sequicity, simply applying original CopyNet architecture is insufficient, since B_t and R_t have different distributions. We here employ two separate RNN (GRU in our implementation) in decoder: one for B_t and the other for R_t. In the first decoding stage, we have a copy-attention mechanism on X to decode B_t; then calculate the generation probability through attending to X as introduced in Sec 3.1, as well as the copy probability for each word $v \in X$ following (Gu et al., 2016) by Eq. 5:

$$P_j^c(v) = \frac{1}{Z} \sum_{i:x_i=v}^{|X|} e^{\psi(x_i)}, j \leqslant m' \tag{5}$$

where Z is a normalization term and $\psi(x_i)$ is the score of "copying" word x_i and is calculated by:

$$\psi(x_i) = \sigma(\mathbf{h}_i^{(x)^T} \mathbf{W}_c) \mathbf{h}_j^{(y)}, j \leqslant m' \tag{6}$$

where $\mathbf{W}_c \in \mathbb{R}^{d \times d}$.

In the second decoding stage (i.e., decoding R_t), we apply the last hidden state of B_t as the initial hidden state of the R_t GRU. However, as we need to explicitly model the dependency on B_t, we have copy-attention mechanism on B_t instead of on X: treating all tokens of B_t as the candidate for copying and attention. Specifically, we use hidden state generated by B_t GRU, i.e., $\mathbf{h}_1^{(y)}, ..., \mathbf{h}_{m'}^{(y)}$, to calculate copying using Eqs. 7 and 8 and attention score as introduced in Sec 3.1. It helps to reduce search space because all key information of X for task completion has been included in B_t.

$$P_j^c(v) = \frac{1}{Z} \sum_{i:y_i=v} e^{\psi(y_i)}, i \leqslant m' < j \leqslant m \tag{7}$$

$$\psi(y_i) = \sigma(\mathbf{h}_i^{(y)^T} \mathbf{W}_c) \mathbf{h}_j^{(y)}, i \leqslant m' < j \leqslant m \tag{8}$$

In contrast to recent work (Eric and Manning, 2017a) that also employs a copy-attention mechanism to generate a knowledge-base search API and machine responses, our proposed method advances in two aspects: on one hand, bspans reduce the search space from $U_1 R_1 ... U_t R_t$ to $B_{t-1} R_{t-1} U_t$ by compressing key points for the task completion given past dialogues; on the other hand, because bspans revisit context by only handling the B_t with a fixed length, the time complexity of TSCP is only $O(T)$, comparing $O(T^2)$ in (Eric and Manning, 2017a).

Involving $\mathbf{k}_t$ when decoding R_t. As $\mathbf{k}_t$ has three possible values: obtaining only one, multiple or no entities. We let $\mathbf{k}_t$ be a vector of three dimensions, one of which signals a value. We append $\mathbf{k}_t$ to the embeddings $\mathbf{y}_j$, as shown in Eq. (9) that is fed into an GRU for generating $\mathbf{h}_{j+1}^{(y)}$. This approach is also referred to as Language Model Type condition (Wen et al., 2016b)

$$\mathbf{y}'_j = \begin{bmatrix} \mathbf{y}_j \\ \mathbf{k}_t \end{bmatrix}, j \in [m'+1, m] \tag{9}$$

4.4 Training

The standard cross entropy is adopted as our objective function to train a language model:

$$\sum_{j=1}^{m} y_j log P_j(y_j) \tag{10}$$

In response generation, every token is treated equally. However, in our case, tokens for task completion are more important. For example, when a user asks for the address of a restaurant, it matters more to decode the placeholder `<address>` than decode words for language fluency. We can employ reinforcement learning to fine tune the trained response decoder with an emphasis to decode those important tokens.

Inspired by (Wen et al., 2017a), in the context of reinforcement learning, the decoding network can be viewed as a policy network, denoted as $\pi_\Theta(y_j)$ for decoding y_j ($m' + 1 \leqslant j \leqslant$ m). Accordingly, the choice of word y_j is an action and its hidden vector generated by decoding GRU is the corresponding state. In reinforcement tuning stage, the trained response decoder is the initial policy network. By defining a proper reward function $r^{(j)}$ for decoding y_j, we can update the trained

Dataset	Cam676		
Size	Train:408 / Test: 136 / Dev: 136		
Domains	restaurant reservation		
Slot types	price, food style etc.		
Distinct slot values	99		
Dataset	KVRET		
Size	Train:2425 / Test: 302 / Dev: 302		
Domains	calendar	weather info.	POI
Slot types	date, etc.	location, etc.	poi, etc.
Distinct slot values	79	65	140

Table 1: Dataset demographics. Following the respective literature, Cam676 is split 3:1:1 and KVRET is split 8:1:1, into training, developing and testing sets, respectively.

response model with policy gradient:

$$\frac{1}{m-m'}\sum_{j=m'+1}^{m} r^{(j)}\frac{\partial log\pi_\Theta(y_j)}{\partial\Theta} \quad (11)$$

where $r^{(j)} = r^{(j)} + \lambda r^{(j+1)} + \lambda^2 r^{(j+2)} + ... + \lambda^{m-j+1} r^{(m)}$. To encourage our generated response to answer the user requested information but avoid long-winded response, we set the reward at each step $r^{(j)}$ as follows: once the placeholder of requested slot has been decoded, the reward for current step is 1; otherwise, current step's reward is -0.1. λ is a decay parameter. Sec 5.2 for λ settings.

5 Experiments

We assess the effectiveness of Sequicity in three aspects: the task completion, the language quality, and the efficiency. The evaluation metrics are listed as follows:

· **BLEU** to evaluate the language quality (Papineni et al., 2002) of generated responses (hence top-1 candidate in (Wen et al., 2017b)).
· **Entity match rate** evaluates task completion. According to (Wen et al., 2017b), it determines if a system can generate all correct constraints to search the indicated entities of the user. This metric is either 0 or 1 for each dialogue.
· **Success F_1** evaluates task completion and is modified from the success rate in (Wen et al., 2017b, 2016a, 2017a). The original success rate measures if the system answered all the requested information (e.g. address, phone number). However, this metric only evaluates recall. A system can easily achieve a perfect task success by always responding all possible request slots. Instead, we here use success F_1 to balance both recall and precision. It is defined as the F_1 score of requested slots answered in the current dialogue.
· **Training time.** The training time is important for iteration cycle of a model in industry settings.

5.1 Datasets

We adopt the CamRest676 (Wen et al., 2017a) and KVRET (Eric and Manning, 2017b) datasets. Both datasets are created by a Wizard-of-Oz (Kelley, 1984) method on Amazon Mechanical Turk platform, where a pair of workers are recruited to carry out a fluent conversation to complete an assigned task (e.g. restaurant reservation). During conversation, both informable and requestable slots are recorded by workers.

CamRest676's dialogs are in the single domain of restaurant searching, while KVRET is broader, containing three domains: calendar scheduling, weather information retrieval and point of interest (POI) Navigation. Detailed slot information in each domain are shown in Table 1. We follow the data splits of the original papers as shown in 1.

5.2 Parameter Settings

For all models, the hidden size and the embedding size d is set to 50. $|V|$ is 800 for CamRes676 and 1400 for KVRET. We train our model with an Adam optimizer (Kingma and Ba, 2015), with a learning rate of 0.003 for supervised training and 0.0001 for reinforcement learning. Early stopping is performed on developing set. In reinforcement learning, the decay parameter λ is set to 0.8. We also use beam search strategy for decoding, with a beam size of 10.

5.3 Baselines and Comparisons

We first compare our model with the state-of-the-art baselines as follow:

- NDM (Wen et al., 2017b). As described in Sec 1, it adopts pipeline designs with a belief tracker component depending on delexicalization.
- NDM+Att+SS. Based on the NDM model, an additional attention mechanism is performed on the belief trackers and a snapshot learning mechanism (Wen et al., 2016a) is adopted.
- LIDM (Wen et al., 2017a). Also based on NDM, this model adopts neural variational inference with reinforcement learning.

	CamRes676					KVRET				
	Mat.	BLEU	Succ. F_1	$Time_{full}$	$Time_{N.B.}$	Mat.	BLEU	Succ. F_1	$Time_{full}$	$Time_{N.B.}$
(1) NDM	0.904	0.212	0.832	91.9 min	8.6 min	0.724	0.186	0.741	285.5 min	29.3 min
(2) NDM + Att + SS	0.904	0.240	0.836	93.7 min	10.4 min	0.724	0.188	0.745	289.7 min	33.5 min
(3) LIDM	0.912	0.246	0.840	97.7 min	14.4 min	0.721	0.173	0.762	312.8 min	56.6 min
(4) KVRN	N/A	0.134	N/A	21.4 min	–	0.459	0.184	0.540	46.9 min	–
(5) TSCP	**0.927**	**0.253**	**0.854**	7.3 min	–	**0.845**	**0.219**	**0.811**	25.5 min	–
(6) Att-RNN	0.851	0.248	0.774	7.2 min	–	0.805	0.208	0.801	23.0 min	–
(7) TSCP$\backslash k_t$	**0.927**	0.232	0.835	7.2 min	–	**0.845**	0.168	0.759	25.3 min	–
(8) TSCP\RL	**0.927**	0.234	0.834	**4.1** min	–	**0.845**	0.191	0.774	**17.5** min	–
(9) TSCP$\backslash B_t$	0.888	0.197	0.809	22.9 min	–	0.628	0.182	0.755	42.7 min	–

Table 2: Model performance on CamRes676 and KVRET. This table is split into two parts: competitors on the upper side and our ablation study on the bottom side. **Mat.** and **Succ. F1** are for match rate and success F1 respectively. $\mathbf{Time_{full}}$ column reports training time till converge. For NDM, NDM+Att+SS and LIDM, we also calculate the training time for the rest parts except for the belief tracker ($\mathbf{Time_{N.B.}}$).

- KVRN (Eric and Manning, 2017b) uses one seq2seq model to generate response as well as interacting with knowledge base. However, it does not incorporate a belief tracking mechanism.

For NDM, NDM+Att+SS, LIDM, we run the source code released by the original authors[2]. For KVRN, we replicate it since there is no source code available. We also performed an ablation study to examine the effectiveness of each component.

- TSCP$\backslash k_t$. We removed the conditioning on k_t when decoding R_t.
- TSCP\RL. We removed reinforcement learning which fine tunes the models for response generation.
- Att-RNN. The standard seq2seq baseline as described in the preliminary section (See §3.1).
- TSCP$\backslash B_t$. We removed bspans for dialogue state tracking. Instead, we adopt the method in (Eric and Manning, 2017a): concatenating all past utterance in a dialogue into a CopyNet to generate user information slots for knowledge base search as well as machine response.

5.4 Experimental Results

As shown in Table 2, TSCP outperforms all baselines (Row 5 vs. Rows 1–4) in task completion (entity match rate, success F1) and language quality (BLEU). The more significant performance of TSCP in KVRET dataset indicates the scalability of TSCP. It is because KVRET dataset has significant lexical variety, making it hard to perform delexicalization for Wen et al.'s model (Rows 1–3)[3]. However, CamRes676 is relatively small with simple patterns where all systems work well. As predicted, KVRN (Row 4) performs worse than TSCP (Row 5) due to lack of belief tracking.

Compared with Wen et al.'s models (Rows 1–3), TSCP takes a magnitude less time to train. Although TSCP is implemented in PyTorch while Wen et al.'s models in Theano, such speed comparison is still valid, as the rest of the NDM model — apart from its belief tracker — has a comparable training speed to TSCP (7.3 mins vs. 8.6 mins on CamRes676 and 25.5 mins vs. 29.3 mins on KVRET), where model complexities are similar. The bottleneck in the time expense is due to belief tracker training. In addition, Wen et al.'s models perform better at the cost of more training time (Rows 1, 2 and 3), suggesting the intrinsic complexity of pipeline designs.

Importantly, ablation studies validate the necessity of bspans. With bspans, even a standard seq2seq model (Att-RNN, Row 6) beats sophisticated models such as attention copyNets (TSCP$\backslash B_t$, Row 9) in KVRET. Furthermore, TSCP (Row 5) outperforms TSCP$\backslash B_t$ (Row 9) in all aspects: task completion, language quality and training speed. This validate our theoretical analysis in Sec 4.3. Other components of TSCP are also important. If we only use vanilla Attention-based RNN instead of copyNet, all metrics for model effectiveness decrease, validating our hypothesize that the copied words need to be specifically modeled. Secondly, BLEU score is sensitive to knowl-

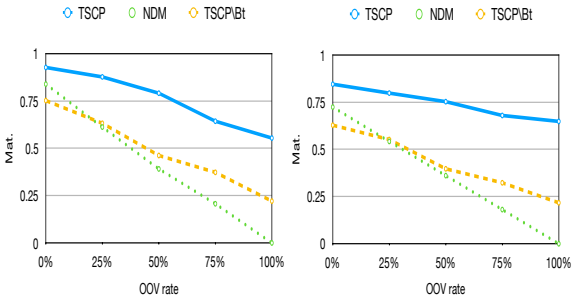
[2]https://github.com/shawnwun/NNDIAL

[3]We use the delexicalization lexicon provided by the original author of KVRET(Eric and Manning, 2017b)

edge base search result $\mathbf{k}_t$ (Row 7 vs. Row 5). By examining error cases, we find that the system is likely to generate common sentences like "you are welcome" regardless of context, due to corpus frequency. Finally, reinforcement learning effectively helps both BLEU and success F_1 although it takes acceptable additional time for training.

5.5 OOV Tests

Previous work predefines all slot values in a belief tracker. However, a user may request new attributes that has not been predefined as a classification label, which results in an *entity mismatch*. TSCP employs copy mechanisms, gaining an intrinsic potential to handle OOV cases. To conduct the OOV test, we synthesize OOV test instances by adding a suffix `unk` to existing slot fillers. For example, we change "I would like Chinese food" into "I would like Chinese_unk food." We then randomly make a proportion of testing data OOV and measure its entity match rate. For simplicity, we only show the three most representative models pre-trained in the in-vocabulary data: TSCP, TSCP$\backslash B_t$ and NDM.



(a) CamRes676 (b) KVRET

Figure 2: OOV tests. 0% OOV rate means no OOV instance while 100% OOV rate means all instances are changed to be OOV.

Compared with NDM, TSCP still performs well when all slot fillers are unknown. This is because TSCP actually learns sentence patterns. For example, CamRes676 dataset contains a frequent pattern "I would like `[food_type]` food" where the `[food_type]` should be copied in B_t regardless what exact word it is. In addition, the performance of TSCP$\backslash B_t$ decreases more sharply than TSCP as more instances set to be OOV. This might be because handling OOV cases is much harder when search space is large.

5.6 Empirical Model Complexity

Traditional belief trackers like (Wen et al., 2017b) are built as a multi-class classifier, which models each individual slot and its corresponding values, introducing considerable model complexities. This is especially severe in large datasets with a number of slots and values. In contrast, Sequicity reduces such a complex classifier to a language model. To compare the model complexities of two approaches, we empirically measure model size. We split KVRET dataset by their domains, resulting in three sub-datasets. We then accumulatively add the sub-datasets into training set to examine how the model size grows. We here selectively present TSCP, NDM and its separately trained belief tracker, since Wen et al.'s set of models share similar model sizes.

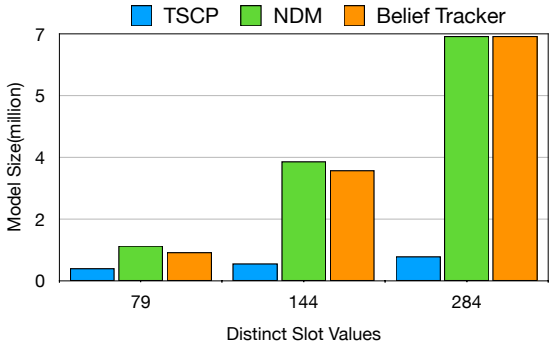


Figure 3: Model size sensitivity with respect to KVRET. Distinct slot values of 79, 144, 284 correspond to the number of slots in KVRET's *calendar*, *calendar + weather info.*, and all 3 domains.

As shown in Figure 3, TSCP has a magnitude less number of parameters than NDM and its model size is much less sensitive to distinct slot values increasing. It is because TSCP is a seq2seq language model which has a approximate linear complexity to vocabulary size. However, NDM employs a belief tracker which dominates its model size. The belief tracker is sensitive to the increase of distinct slot values because it employs complex structures to model each slot and corresponding values. Here, we only perform empirical evaluation, leaving theoretically complexity analysis for future works.

5.7 Discussions

In this section we discuss if Sequicity can tackle *inconsistent user requests* , which happens when users change their minds during a dialogue. Inconsistent user requests happen frequently and are dif-

ficult to tackle in belief tracking (Williams, 2012; Williams et al., 2013). Unlike most of previous pipeline-based work that explicitly defines model actions for each situation, Sequicity is proposed to directly handle various situations from the training data with less manual intervention. Here, given examples about restaurant reservation, we provide three different scenarios to discuss:

- **A user totally changes his mind**. For example, the user request a Japanese restaurant first and says "I dont want Japanese food anymore, I'd like French now." Then, all the slot activated before should be invalid now. The slot annotated for this turn is only *French*. Sequicity can learn this pattern, as long as it is annotated in the training set.

- **User requests cannot be found in the KB** (e.g., *Japanese food*). Then the system should respond like "Sorry, there is no Japanese food...". Consequently, the user can choose a different option: "OK, then French food." The activated slot *Japanese* will be replaced as *French*, which our system can learn. Therefore, an important pattern is the machine-response (e.g., "there is no [XXX constraint]") in the immediate previous utterance.

- **Other cases.** Sequicity is expected to generate both slot values in a belief span if it doesn't know which slot to replace. To maintain the belief span, we run a simple post-processing script at each turn, which detects whether two slot values have the same slot name (e.g., `food_type`) in a pre-defined slot name-value table. Then, such script only keeps the slot value in the current turn of user utterance. Given this script, Sequicity can accurately discover the slot requested by a user in each utterance. However, this script only works when slot values are pre-defined. For inconsistent OOV requests, we need to build another classifier to recognize slot names for slot values.

To sum up, Sequicity, as a framework, is able to handle various inconsistent user input despite its simple design. However, detailed implementations should be customized depends on different applications.

6 Conclusion

We propose Sequicity, an extendable framework, which tracks dialogue believes through the decoding of novel text spans: belief spans. Such belief spans enable a task-oriented dialogue system to be holistically optimized in a single seq2seq model. One simplistic instantiation of Sequicity, called Two Stage CopyNet (TSCP), demonstrates better effectiveness and scalability of Sequicity. Experiments show that TSCP outperforms the state-of-the-art baselines in both task accomplishment and language quality. Moreover, our TSCP implementation also betters traditional pipeline architectures by a magnitude in training time and adds the capability of handling OOV. Such properties are important for real-world customer service dialog systems where users' inputs vary frequently and models need to be updated frequently. For our future work, we will consider advanced instantiations for Sequicity, and extend Sequicity to handle unsupervised cases where information and requested slots values are not annotated.

Acknowledgments

We would like to thank the anonymous reviewers for their detailed comments and suggestions for this paper. This work is also supported by the National Research Foundation, Prime Ministers Office, Singapore under its IRC@SG Funding Initiative.

References

Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. 2017. A survey on dialogue systems: Recent advances and new frontiers. *arXiv preprint arXiv:1711.01731* .

Heriberto Cuayáhuitl, Simon Keizer, and Oliver Lemon. 2015. Strategic dialogue management via deep reinforcement learning. *arXiv preprint arXiv:1511.08099* .

Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. 2017. Towards end-to-end reinforcement learning of dialogue agents for information access. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 484–495.

Mihail Eric and Christopher D Manning. 2017a. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue.

Mihail Eric and Christopher D Manning. 2017b. Key-value retrieval networks for task-oriented dialogue. *SIGDIAL* .

Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. 2017. A convolutional encoder model for neural machine translation. *ACL* .

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *ACL* .

Homa B Hashemi, Amir Asiaee, and Reiner Kraft. 2016. Query intent detection using convolutional neural networks. In *International Conference on Web Search and Data Mining, Workshop on Query Understanding*.

Matthew Henderson, Blaise Thomson, and Jason D Williams. 2014a. The second dialog state tracking challenge. In *SIGDIAL Conference*. pages 263–272.

Matthew Henderson, Blaise Thomson, and Jason D Williams. 2014b. The third dialog state tracking challenge. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, pages 324–329.

Matthew Henderson, Blaise Thomson, and Steve Young. 2013. Deep neural network approach for the dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*. pages 467–471.

Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pages 3128–3137.

John F Kelley. 1984. An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems (TOIS)* 2(1):26–41.

Seokhwan Kim, Luis Fernando DHaro, Rafael E Banchs, Jason D Williams, and Matthew Henderson. 2017. The fourth dialog state tracking challenge. In *Dialogues with Social Robots*, Springer, pages 435–449.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization.

Sungjin Lee. 2013. Structured discriminative model for dialog state tracking. In *SIGDIAL Conference*. pages 442–451.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2015. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055* .

Bing Liu and Ian Lane. 2017. Iterative policy learning in end-to-end trainable task-oriented neural dialog models. *arXiv preprint arXiv:1709.06136* .

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. 2016. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *NAACL*.

Nikola Mrkšić, Diarmuid O Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. 2017. Neural belief tracker: Data-driven dialogue state tracking. *ACL* .

Tim Paek and Eric Horvitz. 2000. Conversation as action under uncertainty. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pages 455–464.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pages 311–318.

Yangyang Shi, Kaisheng Yao, Le Tian, and Daxin Jiang. 2016. Deep lstm based feature mapping for query classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. pages 1501–1511.

Joao Silva, Luísa Coheur, Ana Cristina Mendes, and Andreas Wichert. 2011. From symbolic to sub-symbolic information in question classification. *Artificial Intelligence Review* 35(2):137–154.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* .

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, 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 Gasic, Nikola Mrkšić, Lina M. Rojas Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. 2016b. Conditional generation and snapshot learning in neural dialogue systems. In *EMNLP*. ACL, Austin, Texas, pages 2153–2162. https://aclweb.org/anthology/D16-1233.

Tsung-Hsien Wen, Yishu Miao, Phil Blunsom, and Steve Young. 2017a. Latent intention dialogue models. *ICML* .

Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, 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. *EACL* .

Jason Williams, Antoine Raux, Deepak Ramachandran, and Alan Black. 2013. The dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*. pages 404–413.

Jason D Williams. 2012. A belief tracking challenge task for spoken dialog systems. In *NAACL-HLT Workshop on future directions and needs in the spoken dialog community: tools and data*. Association for Computational Linguistics, pages 23–24.

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. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* .

Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. 2010. The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Computer Speech & Language* 24(2):150–174.

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

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

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*. pages 2852–2858.