

GSN: A Graph-Structured Network for Multi-Party Dialogues

Wenpeng Hu^{1,3,*}, Zhangming Chan^{2,3,*},

Bing Liu^{4,†}, Dongyan Zhao^{2,3}, Jinwen Ma¹ and Rui Yan^{2,3,†}

¹Department of Information Science, School of Mathematical Sciences, Peking University

²Center for Data Science, Peking University

³Institute of Computer Science and Technology, Peking University

⁴Department of Computer Science, University of Illinois at Chicago

{wenpeng.hu,zhangming.chan,zhaody,ruiyan}@pku.edu.cn, liub@uic.edu, jwma@math.pku.edu.cn

Abstract

Existing neural models for dialogue response generation assume that utterances are sequentially organized. However, many real-world dialogues involve multiple interlocutors (i.e., multi-party dialogues), where the assumption does not hold as utterances from different interlocutors can occur “in parallel.” This paper generalizes existing sequence-based models to a *Graph-Structured neural Network* (GSN) for dialogue modeling. The core of GSN is a graph-based encoder that can model the information flow along the graph-structured dialogues (two-party sequential dialogues are a special case). Experimental results show that GSN significantly outperforms existing sequence-based models.

1 Introduction

Most existing dialogue systems are sequence-to-sequence (seq2seq) models [Luan *et al.*, 2016; Serban *et al.*, 2016]. Since a dialogue generally lasts for several turns, a dialogue session with multiple utterances can often be modeled as a sequence of “sequences” (i.e., utterances). A representative framework is the hierarchical recurrent encoder-decoder framework HRED [Serban *et al.*, 2016; Serban *et al.*, 2017]. In HRED, a recurrent neural network (RNN) encoder encodes the tokens in each utterance, and a context RNN encodes the temporal structure of the utterances. The entire dialogue session is then organized as a sequence.

Although HRED is effective in modeling sequential dialogue sessions, it falls short for dialogues involving more than two interlocutors. Table 1 shows a real conversation of 3 people (p_i) in the Ubuntu forum. Utterances 3 and 4 both respond to utterance 2, represented as a graph in Figure 1. We see that utterances can occur in parallel with each other. This is beyond the expressive power of sequence models. This paper generalizes sequence-based representation of two-party dialogues to a graph-based representation of multi-party dialogues. Two-party sequence representation is a special case.

The proposed model, called GSN (*graph-structured network*), models the information flow in a graph-structured di-

| | |
|------------------------|--|
| utterance 1 (p_1): | When the screen goes blank and won't display any login page. |
| utterance 2 (p_2): | I don't know if it's a hardware problem or an os. |
| utterance 3 (p_1): | Did you do any upgrade recently? |
| utterance 4 (p_3): | If it works for one user it's probably not a hardware issue. |

Table 1: A real conversation in the Ubuntu forum.

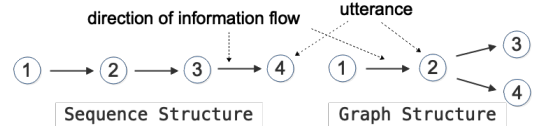


Figure 1: Sequence and graph structures.

alogue. It is a general model and works well for both graph-structured (multi-party) and sequential (two-party) dialogues.

The core of GSN is an *utterance-level graph-structured encoder* (UG-E), which encodes utterances based on the graph topology rather than the sequence of their appearances. Encoding in UG-E is an iterative process. In each iteration, each utterance (a node in the graph) i accepts information from all its preceding utterances (nodes) j . UG-E is thus a generalization of existing sequential encoders, and can handle both sequential and graph-based dialogues.

GSN also models the speaker information as the utterances from the same speaker often have certain relationships. Sequence-based methods in [Li *et al.*, 2016; Zhang *et al.*, 2018b] also learn a user embedding and concatenate it to the utterances. However, GSN builds implicit connections between utterances from the same interlocutor to model the dynamic information flow among his/her utterances with no explicit user representation, which results in performance gains.

In summary, this paper makes the following contributions. (1) It proposes a novel graph-structured network (GSN) to model graph-structured dialogues. Sequence models are a special case. The core of GSN is an utterance-level graph-structured encoder (UG-E). To our knowledge, no work on graph-based representation learning has been done for dialogues. (2) It formulates the linkage within the graph to model users across dialogue sessions. Experiments show that GSN can reach up to 13.85 BLEU points and improve the state-of-the-art baselines by 2.27 (over 16%) BLEU points.

*Equal Contribution.

†Contact Author

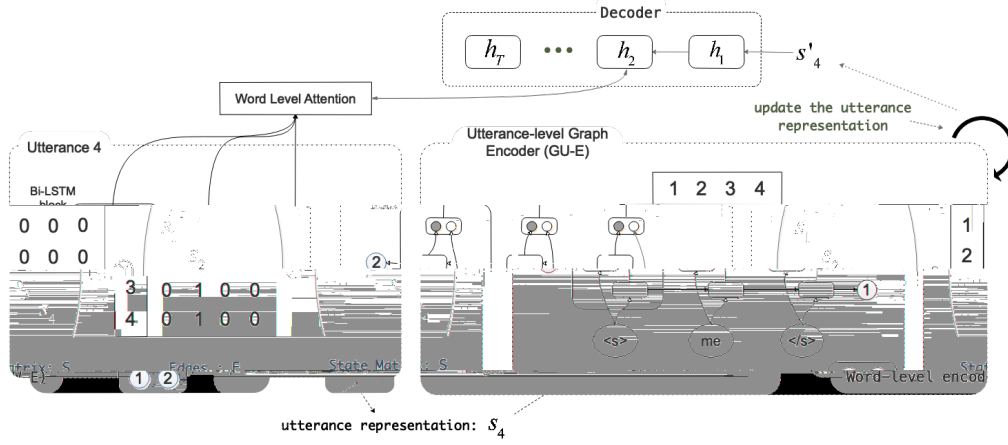


Figure 2: Architecture of GSN.

2 Problem Formulation

Utterances in a structured dialogue session can be formulated as a directed graph $G(V, E)$, where V is a set of m vertices $\{1, \dots, m\}$ and $E = \{e_{i,j}\}_{i,j=1}^m$ is a set of directed edges. Each vertex i is an utterance represented as a vector s_i learned by an RNN. If utterance j is a response to utterance i , then there is an edge from i to j with $e_{i,j} = 1$; otherwise $e_{i,j} = 0$. Our goal is to generate the (best) response \bar{r} that maximizes the conditional likelihood given the graph G :

$$\bar{r} = \arg \max_r \log P(r|G) = \arg \max_r \sum_{i=1}^{|r|} \log P(r_i|G, r_{<i}) \quad (1)$$

where $P(r|G)$ is modeled with the proposed GSN.

This model can be further enhanced by considering the speaker information, which introduces an adjacency matrix $U = \{u_{i,j}\}_{i,j=1}^m$, with $u_{i,j} = 1$ if utterances i and j are from the same speaker and j is after i ; $u_{i,j} = 0$ otherwise.

3 Graph-Structured Neural Network (GSN)

Figure 2 gives the overall framework of GSN, which has three main components: a *word-level encoder* (W-E), an *utterance-level graph-structured encoder* (UG-E), and a decoder. UG-E is the core of GSN. To make Figure 2 concise, we omitted some connecting lines and attentions. ‘ \otimes ’ is a special multiplication operator, called the *update operator* (see below). ‘ \cdot ’ denotes the mathematical matrix multiplication.

3.1 Word-level Encoder (W-E)

Given an utterance $i = (w_{i,1}, w_{i,2}, \dots, w_{i,n})$, W-E encodes it into an internal vector representation. We use a bidirectional recurrent neural network (RNN) with LSTM units to encode each word $w_{i,t}$, $t \in \{1, \dots, n\}$ as a hidden vector $s_{i,t}$:

$$\begin{aligned} \vec{s}_{i,t} &= \overrightarrow{LSTM}(e \cdot w_{i,t}, \vec{s}_{i,t-1}) \\ \overleftarrow{s}_{i,t} &= \overleftarrow{LSTM}(e \cdot w_{i,t}, \overleftarrow{s}_{i,t-1}) \end{aligned} \quad (2)$$

where $e \cdot w_{i,t}$ is the embedding of word $w_{i,t}$ at time step t , $\vec{s}_{i,t}$ is the hidden state for the forward pass LSTM and $\overleftarrow{s}_{i,t}$ for the

backward pass. We use their concatenation, i.e., $[\vec{s}_{i,t}; \overleftarrow{s}_{i,t}]$, as the hidden state $s_{i,t}$ at time t . Note that each word in the utterance indicates a state and a time step.

After encoding by W-E, a session with utterances $\{1, \dots, m\}$ is represented with $S = \{s_i, i \in \{1, \dots, m\}\}$, where $s_i = s_{i,n}$ is the last hidden state of W-E.

3.2 Utterance-level Graph-Structured Encoder (UG-E)

The HRED model is a hierarchical sequence-based word and utterance-level RNN. It predicts the hidden state of each utterance at time step t by encoding the sequence of all utterances appeared so far. Due to graph structures in real dialogues, RNN is no longer suitable for modeling the information flow of utterances. For instance, in Figure 1, HRED cannot handle utterances 3 and 4 properly because they are not logically sequential, but are “in parallel.” The UG-E comes to help.

UG-E & Information Flow Over Graph

To model a graph structure and its information flow, we propose a new RNN with dynamic iterations. Given a session S , only the information in the preceding nodes/vertices i' of each node i will flow to i in an iteration (i.e., there is a directed edge from each i' to i). Then the state of node (utterance) i is updated and the updated state is used in the next iteration. In each iteration, all updates in a session are done in parallel. In this way, the encoding information and gradients can flow fully over the graph after some iterations. For instance, in the session in Figure 1 (or 4(b)), although the information flow of one iteration is from one node’s preceding nodes to the node, the information in 1 can flow to 3 after two iterations.

UG-E’s basic operation is illustrated in Figure 3. For example, given a session $S = (s_1, s_2, s_3, s_4)$, in the l -th iteration, the state of the i -th utterance can be calculated by:

$$\begin{aligned} s_i^l &= s_i^{l-1} + \eta \cdot \Delta s_{I|i}^{l-1} \\ \Delta s_{I|i}^{l-1} &= \sum_{i' \in \varphi} \Delta s_{i'|i}^{l-1} \end{aligned} \quad (3)$$

where φ is the collection of preceding nodes of the current node i in the direction of the information flow; $\Delta s_{I|i}^{l-1}$ is the

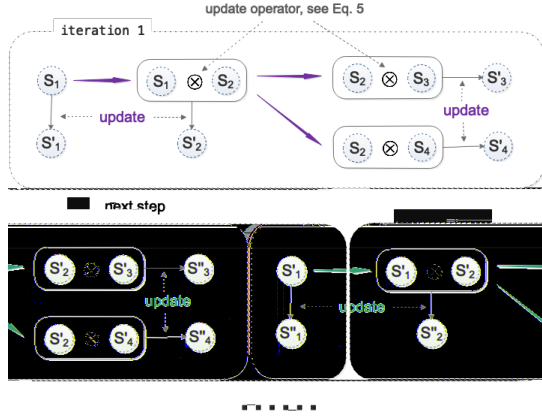


Figure 3: Information flow over the graph.

updating information, which is calculated by Eq. 5 below; η is the updating coefficient indicating how much the new information (from the preceding nodes) should be added to the current state of the i -th utterance (node). Inspired by [Sabour *et al.*, 2017], we design an alpha-weight as the updating coefficient. We use a non-linear ‘squashing’ function (i.e., $\text{SQH}(\cdot)$) to give vectors with a small norm a weight close to α , but a large norm a weight close to 1:

$$\eta = \text{SQH}(\Delta s_{I|i}^{l-1}) = \frac{\alpha + \|\Delta s_{I|i}^{l-1}\|}{1 + \|\Delta s_{I|i}^{l-1}\|} \quad (4)$$

where $\alpha > 0$ is a hyperparameter (it should be greater than 0 to provide enough updating rate from the very beginning); $\Delta s_{I|i}^{l-1}$ is the updating information and is produced based on the state of the current utterance s_i^{l-1} and the state of the preceding utterance $s_{i'}^{l-1}$:

$$\Delta s_{i'|i}^{l-1} = s_{i'}^{l-1} \otimes s_i^{l-1} \quad (5)$$

where ‘ \otimes ’, the *update operator*, computes the updating information. Inspired by the updating operation hidden in Gated Recurrent Units (GRU) [Cho *et al.*, 2014], \otimes is defined as:

$$\begin{aligned} \Delta s_{i'|i}^{l-1} &= (1 - \mathbf{r}_i) * s_{i'}^{l-1} + \mathbf{r}_i * \tilde{\mathbf{h}}_i \\ \tilde{\mathbf{h}}_i &= \tanh(\mathbf{W} \cdot [\mathbf{r}_i * s_{i'}^{l-1}, s_i^{l-1}]) \\ \mathbf{r}_i &= \sigma(\mathbf{W}_x \cdot [s_{i'}^{l-1}, s_i^{l-1}]) \\ \mathbf{r}_i &= \sigma(\mathbf{W}_r \cdot [s_{i'}^{l-1}, s_i^{l-1}]) \end{aligned} \quad (6)$$

where \mathbf{W} , \mathbf{W}_x and \mathbf{W}_r are parameters to be learned. σ is the sigmoid function.

Bi-directional Information Flow

In Figure 4(a), utterances 3 and 4 are two responses to utterance 2. It is obvious that utterance 2 can help generate a better state for utterance 4 and vice versa. However, the algorithm introduced above only allows the information and gradients to flow over the forward direction of the graph (as shown by the purple arrows in Figure 4(a)). Hence the information in utterance 3 cannot flow to utterance 4.

To tackle this problem, we propose a Bi-directional Information Flow (BIF) algorithm, which also uses backward

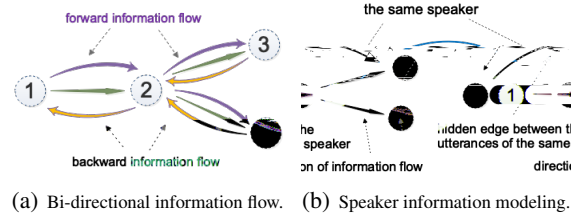


Figure 4: Information flow.

information flow (as shown by the orange arrows in Figure 4(a)). In order to allow information to flow thoroughly, we push the information to flow backward first and then forward to ensure that the information can flow from one node to one’s sibling nodes, i.e., backward to parent and forward to siblings. In our example above, the information of utterance 3 can flow to utterance 4 through utterance 2 after one backward flow and one forward flow, illustrated in Figure 4(a).

Speaker Information Flow

Representing speaker information in the latent embedding space is a popular method to enhance dialogue generation. However, this method lacks the ability to model the speaker and the dynamic changes of the speaker’s ideas in a given session, especially when the speaker only speaks a few times because there may not be enough data to train those embeddings to represent the speaker and the changes since this method usually requires large data to train [Li *et al.*, 2016; Qian *et al.*, 2017; Zhang *et al.*, 2018b].

Since the changes in speaker utterances reflect the changes in his/her mind, we propose to create an edge for every pair of utterances from the same speaker following the chronological order of the utterances. Thus there should be hidden edges among all utterances of the same user (e.g., the edge from utterances 1 to 3 in Figure 4(b)). We employ the same \otimes operation to process the hidden edges, but due to different parameters, we use \otimes to denote it:

$$\Delta s_{i'|i}^{l-1} = s_{i'}^{l-1} \otimes s_i^{l-1} \quad (7)$$

We add the speaker information to Eq. 3:

$$\begin{aligned} s_i^l &= s_i^{l-1} + \eta \cdot \Delta s_{I|i}^{l-1} + \lambda \cdot \Delta s_{I|i}^{l-1} \\ \Delta s_{I|i}^{l-1} &= \sum_{i' \in \varphi} \Delta s_{i'|i}^{l-1} \end{aligned} \quad (8)$$

where η and $\Delta s_{I|i}^{l-1}$ are the same as those in Eq. 3; λ is also calculated with Eq. 4 with input $\Delta s_{I|i}^{l-1}$ instead of $\Delta s_{I|i}^{l-1}$.

3.3 Reformulation as Matrix Operations

So far, we have presented the proposed model. For computation, we reformulate it as matrix operations (also see UG-E in Figure 2) and give the pseudo-code in Algorithm 1. Recall the session $\mathbf{S} = (s_1, s_2, s_3, s_4)$, which is used to build the graph $\mathbf{G}(V, E)$ in Figure 4(a). We build a state matrix \mathbf{S} with the vertices of the graph \mathbf{G} (also the session \mathbf{S}) as the diagonal elements, and all the other elements are set to 0 (we name this process the *Building State Matrix function*, denoted by

$BSM(S)$). We then use the “@” relation (a speaker responding to another speaker) as the connection between two vertices to build the edge matrix \mathbf{E} (shown in Figure 2). Recall the speaker information modeling in Section 3.2 and the utterance speaker adjacency matrix \mathbf{U} . The main operation of Eq. 8 can be formalized by:

$$\Delta \mathbf{E} = \mathbf{S}^{l-1} \cdot \mathbf{E} \otimes \mathbf{S}^{l-1}; \Delta \mathbf{U} = \mathbf{S}^{l-1} \cdot \mathbf{U} \otimes \mathbf{S}^{l-1} \quad (9)$$

$$\mathbf{S}^l = \mathbf{S}^{l-1} + BSM(\boldsymbol{\eta} \odot \Delta \mathbf{E} + \boldsymbol{\lambda} \odot \Delta \mathbf{U}) \quad (10)$$

where $\Delta \mathbf{E} = \{\Delta \mathbf{E}_{i,j}\}_{i=1}^m$ and $\Delta \mathbf{U} = \{\Delta \mathbf{U}_{i,j}\}_{i=1}^m$ are two vectors, m is the length of the given session; \odot denotes the Hadamard product; $\boldsymbol{\eta}$ and $\boldsymbol{\lambda}$ can be calculated by:

$$\boldsymbol{\eta} = \{\text{SQH}(\Delta \mathbf{E}_i)\}_{i=1}^m; \boldsymbol{\lambda} = \{\text{SQH}(\Delta \mathbf{U}_i)\}_{i=1}^m \quad (11)$$

This is just the forward information flow. We can obtain the backward information flow operation by changing Eq. 9:

$$\Delta \mathbf{E} = \mathbf{S}^{l-1} \cdot \mathbf{E}^T \otimes \mathbf{S}^{l-1}; \Delta \mathbf{U} = \mathbf{S}^{l-1} \cdot \mathbf{U}^T \otimes \mathbf{S}^{l-1} \quad (12)$$

To obtain $\Delta \mathbf{E}$ and $\Delta \mathbf{U}$ in Eq. 10, we need to change the direction of the sum, i.e., $\Delta \mathbf{E} = \{\Delta \mathbf{E}_{j,i}\}_{i=1}^m$ and $\Delta \mathbf{U} = \{\Delta \mathbf{U}_{j,i}\}_{i=1}^m$. \mathbf{S} , \mathbf{E} , and \mathbf{U} can be very sparse. But the proposed method can be well organized and the sparse matrices can be addressed by sparse matrix operations. The pseudo-code is given in Algorithm 1 in Appendix ¹.

3.4 Decoder

As shown in Figure 2, we illustrate a session $\{i\}_{i=1}^m$ with the corresponding encoding state denoted by \mathbf{S} . To generate a response to an utterance i , the decoder calculates a distribution over the vocabulary and sequentially predicts word r_k using a softmax function:

$$p(\mathbf{r}|\mathbf{S}; \theta) = \prod_{k=1}^{|\mathbf{r}|} P(r_k | \mathbf{S}_{i,i}, \mathbf{r}_{<k}; \theta) = \prod_{k=1}^{|\mathbf{r}|} \text{softmax}(f(\mathbf{h}_k, \mathbf{c}_k, r_{k-1})) \quad (13)$$

where $f(\cdot)$ is the tanh function. r_{k-1} is the word generated at the $(k-1)$ -th time step, obtained from a word look-up table. $\mathbf{h}_k = \text{GRU}(\mathbf{h}_{k-1}, r_{k-1})$ is the hidden state variable of a GRU at time step k . $\mathbf{h}_0 = \mathbf{S}_{i,i}$, and \mathbf{c}_k is the attention-based encoding of utterance i at decoding time step k and it is calculated by $\mathbf{c}_k = \frac{\sum_{j=1}^n \exp(e_{j,k}) \mathbf{s}_{i,j}}{\sum_{j=1}^n \exp(e_{j,k})}$, where $\mathbf{s}_{i,j}$ is the encoder hidden state at time step j for utterance i , and $e_{j,k} = \mathbf{h}_k \mathbf{W}_a \mathbf{s}_{i,j}$ scores the match degree of \mathbf{h}_k and $\mathbf{s}_{i,j}$.

4 Experiments

4.1 Experimental Setups

Data Preparation

Our experiment uses the Ubuntu Dialogue Corpus² [Lowe *et al.*, 2015] as it is the only benchmark corpus with annotated multiple interlocutors. It is also large with almost one million multi-turn dialogues, over seven million utterances and 100 million words. Each record contains a response utterance with its speaker ID and posting time.

To build the training and testing datasets, we extract all utterances with response relations indicated by the “@” symbol in the corpus. For example, “A @ B” means that the utterance is addressed to Speaker B by Speaker A. Utterances from Speaker A and Speaker B are encoded into vector representations and used to construct the state matrix introduced in Section 3.3 as vertices. A directed edge is installed from A to B and used to build the edge matrix described in Sec. 3.3.

Following the baselines, we take the last utterance in each given session as the utterance to be generated³ (i.e., the output target) and the other utterances in the session as the input. Finally, we extracted 380k sessions (about 1.75M utterances) as the experiment corpus and each session has 3 to 10 utterances and 2 to 7 interlocutors. We randomly divide the corpus into the training, development (with 5k q/a pairs), and test (with 5k q/a pairs) sets. We report the results on the test set. In testing, following the graph structure, the system knows which utterances to respond to. This is reasonable as this is also the case in a human dialogue, i.e., before responding, we know which preceding utterances to rely to.

It is important to note here that GSN is a general model that works well for both graph-structured (multi-party) and sequential (two party) party3 (d(de)t250.0059e96 (W)77.98903 (de)25.000

¹https://morning-dews.github.io/Appendix/IJCAI2019_GSN.pdf

²<http://dataset.cs.mcgill.ca/ubuntu-corpus-1.0/>

| Model | BLEU 1 | BLEU 2 | BLEU 3 | BLEU 4 | METEOR | ROUGE _L |
|------------------------------------|--------------------------|-------------------------|-------------------------|-------------------------|-------------------------|--------------------------|
| seq2seq | 10.45 | 4.13 | 2.08 | 1.02 | 3.43 | 9.67 |
| seq2seq W-speaker | 10.70 | 4.98 | 2.20 | 1.55 | 3.92 | 9.42 |
| Seq2seq (last utte) | 9.85 | 3.04 | 1.38 | 0.67 | 3.98 | 8.34 |
| HRED [Serban <i>et al.</i> , 2016] | 10.80 | 4.60 | 2.54 | 1.42 | 4.38 | 10.23 |
| HRED W-speaker | 11.23 | 4.82 | 3.06 | 1.64 | 4.36 | 10.98 |
| GSN No-speaker (1-iter) | 9.42 | 3.05 | 1.61 | 0.95 | 3.74 | 7.63 |
| GSN No-speaker (2-iter) | 12.06 | 4.87 | 2.80 | 1.70 | 4.32 | 10.09 |
| GSN No-speaker (3-iter) | 12.77 ^N | 5.37 ^N | 3.17 | 1.99 ^N | 4.53 | 10.80 |
| GSN W-speaker (1-iter) | 10.31 | 4.06 | 2.34 | 1.45 | 3.88 | 9.96 |
| GSN W-speaker (2-iter) | 12.77 | 4.93 | 2.61 | 1.46 | 4.79 | 11.34 |
| GSN W-speaker (3-iter) | 13.50^N | 5.63^N | 3.24^N | 1.99^N | 4.85^N | 11.36^N |

Table 2: Experimental results, conducted in different settings, including sequential data and graph data using different models based on automated evaluation. ‘Seq2seq (last utte)’ is trained by using only the last utterance before the final response of the session as the input (all utterances before are ignored). ‘*n*-iter’ means that the results are obtained after *n* iterations. ‘No-speaker’ is our proposed GSN model without speaker information flow while ‘W-speaker’ has it. [▲]denotes the *p*-value < 0.01 in paired *t*-test against the best baseline (shaded row).

Benefited from dynamic iterations, GSN is very flexible in generating responses for any utterance in a given session. However, to be consistent with the baselines, only the last utterance in each session is used as the target in training and testing. The code of our model can be found here ⁴.

4.2 Training Details of Our GSN Model

We share the word embedding between the word-level encoder and the decoder and limit the shared vocabulary to 30k. The number of hidden units is set as 300 and the word embedding dimension is set as 300. We have 2 layers for both word-level encoder and decoder. The network parameters are updated using the Adam algorithm [Kingma and Ba, 2014] with the learning rate of 0.0001. All utterances are clipped to 30 words. We run all experiments on a single GTX Titan X GPU, and training takes 25 epochs.

4.3 Results and Analysis

Automated Evaluation

We use two kinds of metrics in automated evaluation: 1) Following [Fu *et al.*, 2017; Havrylov and Titov, 2017], we use the evaluation package of [Chen *et al.*, 2015], which includes BLEU 1 to 4, METEOR and ROUGE_L. 2) We also use embedding-based metrics [Forgues *et al.*, 2014] which can cover the weaknesses of the BLEU’s.

Table 2 shows the evaluation results. The first three rows are for the baselines. The three rows in the middle are for our GSN model using only the information flow over the graph structure, and the last three rows are also for our GSN model but with the addition of the speaker information flow. From Table 2, we can make the following observations:

(1). GSN (row 11, with the speaker information flow after 3 iterations) markedly outperforms the baselines (rows 2 and 5) by up to 2.27 BLEU points (BLEU 1).

(2). With-speaker (W-speaker) versions of GSN also clearly outperform the no-speaker versions, indicating the importance of the speaker information flow. To further verify whether the improvement is due to adding more connections or adding the speaker edges, we conducted experiments by adding some random edges with different percentages until full connections

among nodes (using No-speaker setting with 3 iterations). The results showed a clear drop with the increase of randomly added edges and received very poor result for full connection, which further shows the usefulness of the proposed speaker information modeling method. As the results are very poor, they are not shown here.

(3). We also use the exist embedding-based persona method to arm baselines (W-speaker) for a fair comparison with GSN W-speaker version. We can see from Table 2 the gain is limited, and our method still outperforms the baselines.

(4). The results of GSN improve as the number of iterations increases, which indicates the importance of the dynamic iterations. With more iterations, more utterances will be modeled by GSN. Only after two iterations, our models with the speaker information flow (row 10) already outperforms both two baselines in 4 out of 6 evaluation metrics. Even for the no-speaker versions (row 7), our model beats the best baseline in 4 out of 6 evaluation metrics.

The BLEU scores had a tiny increase in the 4th iteration (around 0.1% for GSN W-speaker, and 0.3 % for GSN No-speaker). For other metrics, e.g., METEOR and ROUGE_L, there is little change. The 5th iteration is similar, but the scores decrease from the 6th iteration. We thus choose 3 iterations.

Embedding-based metrics: Based on the embedding-based metrics, our model also outperforms the baselines. Our model gets 0.770 / 1.040 / 0.651 for the three embedding-based metrics (Embedding Average Score / Embedding Greedy Score / Embedding Extrema Score), which are all better than the scores of the best baseline model (HRED), 0.515 / 0.905 / 0.325. See the details in *Appendix*¹, which also includes a case study with examples.

Sequential data and graph data: To verify the generic nature of the GSN model, we conduct an ablation experiment with only sequential data (sessions with only two interlocutors) or graph data (remaining sessions with more than two interlocutors). The results are shown in Table 3. We see that GSN significantly outperforms the strong HRED baseline in both sequential and graph settings. From both Tables 3 and 2, we can see that GSN improves in the sequential case mainly because of the additional encoding iterations.

Tables 3 and 2 also show that the proposed iterative graph-

⁴<https://github.com/morning-dews/GSN-Dialogues>

| Model | BLEU 1 | BLEU 2 | BLEU 3 | BLEU 4 | METEOR | ROUGE _L |
|---|--------------|-------------|-------------|-------------|-------------|--------------------|
| HRED [Serban <i>et al.</i> , 2016] (sequential) | 9.61 | 3.48 | 1.86 | 1.01 | 4.08 | 8.22 |
| GSN No-speaker (2-iter sequential) | 11.39 | 4.55 | 2.68 | 1.71 | 4.40 | 9.74 |
| GSN W-speaker (1-iter sequential) | 8.69 | 3.1 | 1.78 | 1.19 | 3.67 | 9.19 |
| GSN W-speaker (2-iter sequential) | <u>12.72</u> | 4.84 | 2.59 | 1.59 | <u>4.70</u> | <u>11.41</u> |
| GSN W-speaker (3-iter sequential) | 12.03 | <u>4.92</u> | <u>2.94</u> | <u>1.97</u> | 4.31 | 10.1 |
| HRED [Serban <i>et al.</i> , 2016] (graph) | 12.16 | 4.90 | 2.68 | 1.49 | 4.42 | 10.90 |
| GSN No-speaker (2-iter graph) | 12.35 | 5.17 | 3.08 | 1.81 | 4.43 | 10.42 |
| GSN W-speaker (1-iter graph) | 10.66 | 4.36 | 2.52 | 1.50 | 3.97 | 10.10 |
| GSN W-speaker (2-iter graph) | 12.76 | 5.23 | 2.94 | 1.75 | 4.80 | 11.33 |
| GSN W-speaker (3-iter graph) | 13.85 | 5.83 | 3.33 | 1.98 | 5.10 | 11.66 |

Table 3: Experimental results of using *sequential data* (with only 2 interlocutors, the first five rows of the results) or *graph data* only (with more than 2 interlocutors, the last five rows of the results). The symbol string ‘*n*-iter’, ‘No-speaker’ and ‘W-speaker’ in the table have the same meaning as those in Table 2. The result of GSN No-speaker 3-iter isn’t given as it performs worse.

| Human | HRED | No-speaker | | W-speaker | |
|-------|------|------------|--------|-------------------|-------------------------|
| | | 1-iter | 3-iter | 1-iter | 3-iter |
| 3.01 | 1.91 | 1.89 | 1.98 | 2.23 [▲] | 2.37[▲] |

Table 4: Human evaluation results. [▲]denotes p -value < 0.01 in paired t -test against HRED. The perfect score is 4.

structured encoder UG-E and the graph-based speaker information flow are effective. GSN is thus a good generalization of the sequence-based models, and a desirable system for both graph-structured (multi-party) and sequential (two-party) dialogue response generation.

Human Evaluation

We also conducted human evaluation to measure the quality of responses generated by all methods. We evaluate based on “*naturalness*”, which includes 1) grammaticality, 2) fluency and 3) rationality. We randomly sampled 100 utterance-response pairs, shuffled the order of systems, and asked three Ph.D. students to rate the pairs in terms of model quality on 0 to 4 scales (4 for the best) and we report their average scores. More details can be found in *Appendix*¹.

Table 4 shows that GSN (with the speaker information flow after only 1 iteration or the no-speaker version after 3 iterations) outperforms the best baseline (HRED), indicating that GSN generates more natural responses. The reason that our model (with the speaker information flow and just one iteration, column 5 in Table 4) outperforms three iterations of our model’s no-speaker version (column 4 in Table 4) is because the model (with the speaker information flow) can generate a more consistent response for the speaker. The generated utterances are more preferred by humans.

5 Related Work

Existing dialogue models follow the sequential information flow [Shang *et al.*, 2015; Wen *et al.*, 2017; Tao *et al.*, 2019]. Recent progresses in seq2seq models [Sutskever *et al.*, 2014; Luong *et al.*, 2015] have inspired several efforts [Li *et al.*, 2019; Young *et al.*, 2018] to build dialogue systems.

Although seq2seq models have achieved good results for dialogue generation, they regard all input utterances as a long sequence, which greatly increases the complexity of the model

in passing information and computing gradients. As an improved solution, the HRED models [Serban *et al.*, 2016] tackle this problem by constructing the sequential flow at the utterance level. However, this setting is insufficient for modeling dialogues that have more than 2 interlocutors, which need a graph-based model.

For multi-party dialogues, prior work have employed retrieval-based approaches [Zhang *et al.*, 2018a; Meng *et al.*, 2018]. No graph modeling method has been proposed, although graph-based methods have been used for other NLP tasks, e.g., Graph Convolutional Networks for classification [Kipf and Welling, 2016] and semantic role labeling [Marcheggiani and Titov, 2017], Gated Graph Neural Networks for generation from AMR graphs and syntax-based neural machine translation [Beck *et al.*, 2018]. Different from these works, we propose a generation model by formulating the complex dialogue problem using a graph-based solution. Also importantly, compared to seq2seq and HRED, GSN not only can encode graph structured information flows, but also sequential ones.

6 Conclusion

In this paper, we proposed a *general graph-structured neural network* GSN to model both graph-structured (multi-party) and sequential (two-party) dialogues. The core of the model is an utterance-level graph-based encoder (UG-E), which is a generalization of the conventional sequence-based encoder. For the response generation in multi-party conversations, the speaker information is also modeled in the graph. As our results showed, GSN is general and is suitable for both multi-party and two-party dialogues.

The current GSN relies on clear addressee information. Our future work will try to automatically identify the conversation structure and decide who to respond to. Dynamic routine and attention can be leveraged to achieve this goal.

Acknowledgements

This work was partially supported by National Key Research and Development Program of China (No.2017YFC0804001), National Science Foundation of China (No. U1604153, No. 61876196, No. 61672058), Alibaba Innovative Research Fund. Rui Yan was supported by CCF-Tencent Open Research Fund and Microsoft Research Asia Collaborative Research Program.

[E

- Skripf and Welling, 2016 (arXiv-249.995 80)]TJ66 T02907.963 Tf[(,)-249.995 (2017)]TJ44.83168 -10.95Tj616[