Line Graph Enhanced AMR-to-Text Generation with Mix-Order Graph Attention Networks

Yanbin Zhao, Lu Chen, Zhi Chen, Ruisheng Cao, Su Zhu, Kai Yu*
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
SpeechLab, Department of Computer Science and Engineering
Shanghai Jiao Tong University, Shanghai, China
{zhaoyb, chenlusz, zhenchi713}@sjtu.edu.cn
{211314, paul2204, kai.yu}@sjtu.edu.cn

Task definition & motivation

- AMR-text generation aims to recover nature language from Abstract Meaning Representation
- Existing graph-to-sequence model utilizes graph encoder, it is only guided by the first order adjacency information.
- The relationships between labeled edges are not fully considered.

Framework

- We separate the AMR graph into two sub-graphs without labeled edges– concept graph and line graph

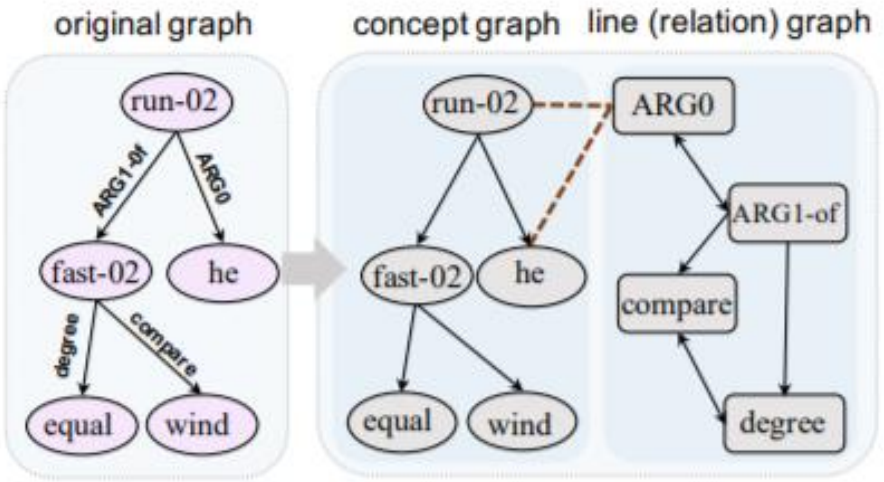

Figure 1: An AMR graph (left) for sentence "He runs as fast as the wind." and its concept graph and relation graph (line graph). Two graphs are aligned with each other based on the node-edge relations in the original graph.

The line graph of a graph G is another graph L(G) that represents the adjacencies between Edges of G. it is defined as:

- Each node of L(G) represents an edge of G
- Two nodes of L(G) are adjacent if and only if

Their corresponding edges share a common node in G

Framework

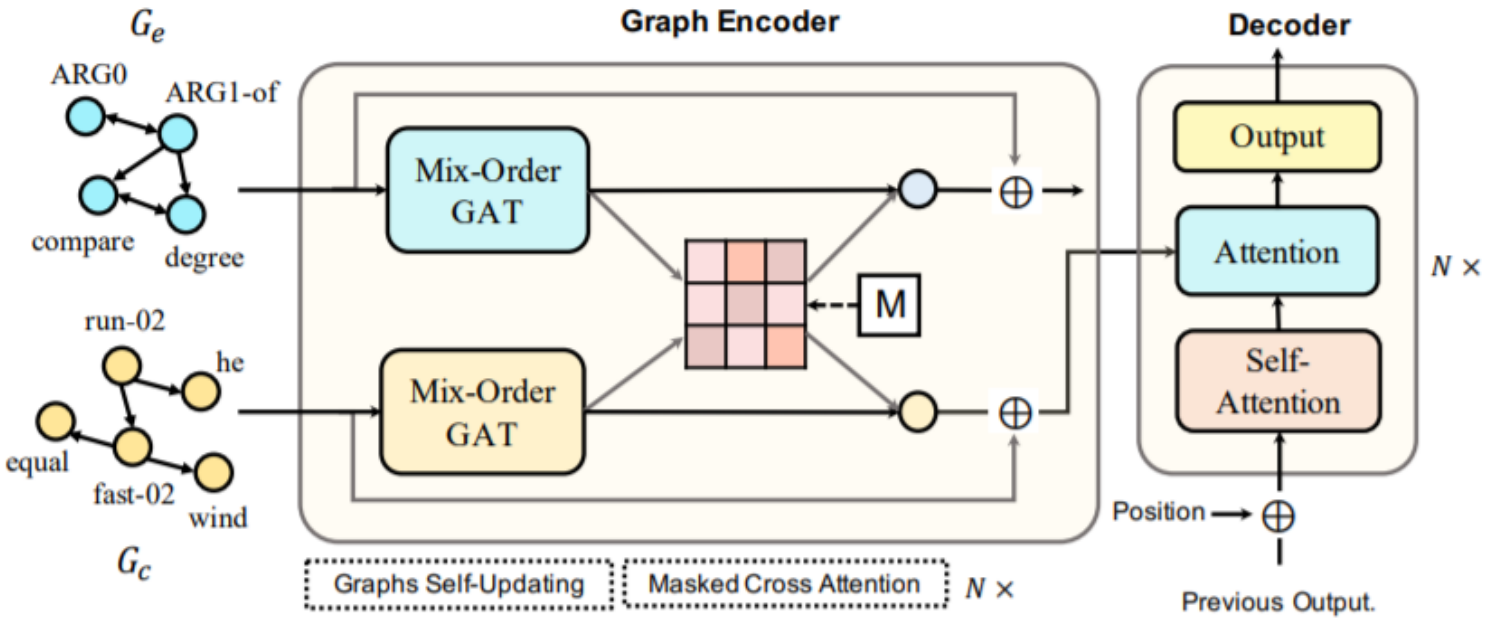

Figure 3: An overview of our proposed model

Model Mix-Order GAT

- The relations between indirectly connected nodes are ignored in a traditional graph attention layer
- Mix Order GAT explores these relations by mixing the higher order neighbourhood information. Let $\mathcal{R}^k(x_i)$ denotes the k-th order neighbourhood,which means all nodes in $\mathcal{R}^k(x_i)$ are reachable for x_i within k hops:

$$\mathcal{R}^k(x_i) = \bigcup_{x_j \in \mathcal{R}^{k-1}(x_i)} \mathcal{N}_+(x_j).$$

At l-th update step, each x_i will interact with its reachable neighbours:

$$\begin{aligned}\mathbf{h}_i^l &= \text{MixGAT}^l(\mathbf{h}_i^{l-1}, \mathbf{R}^K) \\ &= \Big\Vert_{k=1}^{K} \sigma\left(\sum_{x_j \in \mathcal{R}^k(x_i)} \alpha_{ij}^k \mathbf{h}_j^{l-1} \mathbf{W}_k^l\right),\end{aligned}$$

Model self-updating

- Let $\boldsymbol{C_{l-1}} = \left\{\boldsymbol{e_i^{l-1}}\right\}_{\boldsymbol{i=1}}^{\boldsymbol{m}}$ and $\boldsymbol{E_{l-1}} = \left\{\boldsymbol{e_i^{l-1}}\right\}_{\boldsymbol{i=1}}^{\boldsymbol{n}}$ to denote the input embedding of the l-th encoding layer,
- The representations of the two graphs are updated independently by mix-order graph attention networks:

$$\vec{\mathbf{C}}_{self}^{l} = \mathrm{MixGAT}_{c1}^{l}(\mathbf{C}^{l-1}, \mathbf{R}_{c}^{K}),$$
$$\vec{\mathbf{E}}_{self}^{l} = \mathrm{MixGAT}_{e1}^{l}(\mathbf{E}^{l-1}, \mathbf{R}_{e}^{K}).$$

- Note that both G_c and G_e are directed graphs and unidirectional propagation loses the structural information in the reversed direction.

Model-self updating

- We further employ dual graph:Dual graph has the same node representations but reversed edge directions compared to the original graph. If A->B exists in the original graph, then it turns to B->A

  In the dual graph

- Denote $\widetilde{G_c}$ and $\widetilde{G_e}$ as the dual graph of G_c and G_e, respectively. We then have:

$$\breve{\mathbf{C}}^{l}_{self} = \mathrm{MixGAT}^{l}_{c2}(\mathbf{C}^{l-1}, \widetilde{\mathbf{R}}^{K}_{c}),$$
$$\breve{\mathbf{E}}^{l}_{self} = \mathrm{MixGAT}^{l}_{e2}(\mathbf{E}^{l-1}, \widetilde{\mathbf{R}}^{K}_{e}).$$

Model-self updating

- We obtain the final representation by combing bidirectional information:

$$\mathbf{C}_{self}^{l} = \left[\vec{\mathbf{C}}_{self}^{l}; \overleftarrow{\mathbf{C}}_{self}^{l}\right] \mathbf{W}_{c1}^{l},$$
$$\mathbf{E}_{self}^{l} = \left[\vec{\mathbf{E}}_{self}^{l}; \overleftarrow{\mathbf{E}}_{self}^{l}\right] \mathbf{W}_{e1}^{l},$$

Model-masked cross attention

- it is also necessary to explore the dependencies between concept nodes and relation nodes.
- We use a matrix $M \in \mathbb{R}^{n \times m}$ to mask the attention weights of unaligned pairs between G_c and G_e, we let $m_{ij} = 0$ if $y_i \in V_e$ is aligned to $x_j \in V_c$ else $m_{ij} = -\infty$. The matrix of attention weights can be calculated as :

$$\mathbf{A}_l = \left(\mathbf{E}_{self}^l \mathbf{W}_{a1}^l\right)\left(\mathbf{C}_{self}^l \mathbf{W}_{a2}^l\right)^T + \mathbf{M},$$

Model-masked cross attention

- For nodes in E^l_{self} , the relevant representation from C^l_{self} is identified as:

$$\mathbf{E}^l_{cross} = \text{softmax}\left(\mathbf{A}_l\right)\mathbf{C}^l_{self},$$

- The same calculation is performed for nodes in C^l_{self} as:

$$\mathbf{C}^l_{cross} = \text{softmax}(\mathbf{A}_l^T)\mathbf{E}^l_{self}.$$

- The final outputs of a graph encoding layer are the combination of the original embeddings and the context representations from another graph.

$$\mathbf{C}^l = \text{FFN}\left(\left[\mathbf{C}^l_{self}; \mathbf{C}^l_{cross}\right]\mathbf{W}^l_{c2} + \mathbf{C}^{l-1}\right),$$
$$\mathbf{E}^l = \text{FFN}\left(\left[\mathbf{E}^l_{self}; \mathbf{E}^l_{cross}\right]\mathbf{W}^l_{e2} + \mathbf{E}^{l-1}\right),$$

Model -decoder

- The decoder of our system is similar to the Transformer decoder
- Note that the outputs of our graph encoder have two parts:concept representation and relation representation. For generation,concept information is **more** important, so we only use the concept representation $\boldsymbol{C}_N$ as the encoder output in the decoder side.

Experiments

- We conduct our experiments in two datasets:: LDC2015E85 and LDC2017T10. We segment natural words in both AMR graphs and references into sub-words. A word in AMR may be divided into several subword and a special edge “subword” is used to connect them

Experiments

Models	LDC2015E86		LDC2017T10	
	BLEU	Meteor	BLEU	Meteor
Sequence-Based Model				
Seq2Seq (Konstas et al., 2017)	22.00	–	–	–
Syntax+S2S (Cao and Clark, 2019)	23.50	–	26.80	–
Graph-Based Model				
Graph LSTM (Song et al., 2018)	23.30	–	–	–
GCNSEQ (Damonte and Cohen, 2019)	24.40	23.60	24.54	24.07
Dual Graph (Ribeiro et al., 2019)	24.32	30.53	27.87	33.21
DCGCN (Guo et al., 2019)	25.70	31.50	27.60	34.00
Transformer-Based Model				
Transformer (Zhu et al., 2019)	25.50	33.16	27.43	34.62
Graph Transformer (Cai and Lam, 2019)	27.40	32.90	29.80	35.10
Structural Transformer (SA) (Zhu et al., 2019)	29.66	35.45	31.54	36.02
Our Approach				
Line Graph + MixGAT, $K = 1$	28.64	34.51	29.96	35.15
Line Graph + MixGAT, $K = 2$	29.62	35.38	31.06	36.13
Line Graph + MixGAT, $K = 4$	**30.58**	**35.81**	**32.46**	**36.78**

```
(f / feel-02
   :ARG0 (h / he)
   :ARG1 (p / person
      :quant (m / more)
      :ARG0-of (c / compete-01)
      :ARG1-of (n / new-01)
      :source (c2 / country
         :poss (w / we)))
      :ARG0-of (p2 / participate-01
         :ARG1 (c3 / compete-01
            :mod (t / this)))))
```

Reference: he felt that , there were more new competitors from our country participating in this competition .

$\mathbf{K_e = 0}$: he feels more competition from our country who participate in this competition .
$\mathbf{K_e = 4}$: he feels that more new competitors from our country who participate in this competition .

(a)

Recursive Template-based Frame Generation for Task Oriented Dialog

Rashmi Gangadharaiah
AWS AI, Amazon
rgangad@amazon.com

Balakrishnan Narayanaswamy
AWS AI, Amazon
muralibn@amazon.com

Task definition and motivation

- Natural Language Understanding (NLU) processes a user's request and convert it into structured information that can be consumed by downstream tasks.
- This information is typically represented as a semantic frame that captures the intent and slot-labels provided by the user.
- Such a shallow representation is insufficient as it does not capture the recursive nature inherent in many domains.

Task definition and motivation

- Flat structures, The frame consists of intents which capture information about the goal of the user and slot-labels which capture constraints that need to be satisfied

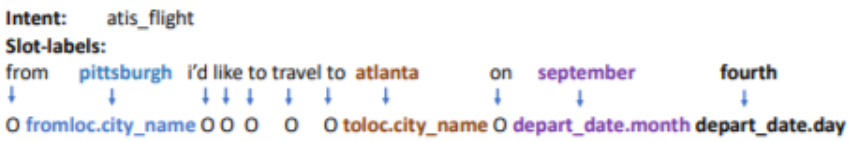

Figure 1: Flat structures used to represent Intents and slot labels in ATIS. 'O' for *Other* or irrelevant tokens.

A novel frame representation

Take “Example: “from pittsburgh i'd like to travel to atlanta on september fourth” as an example, the intent is to book a flight, and the slot labels are the from location, to location and the date

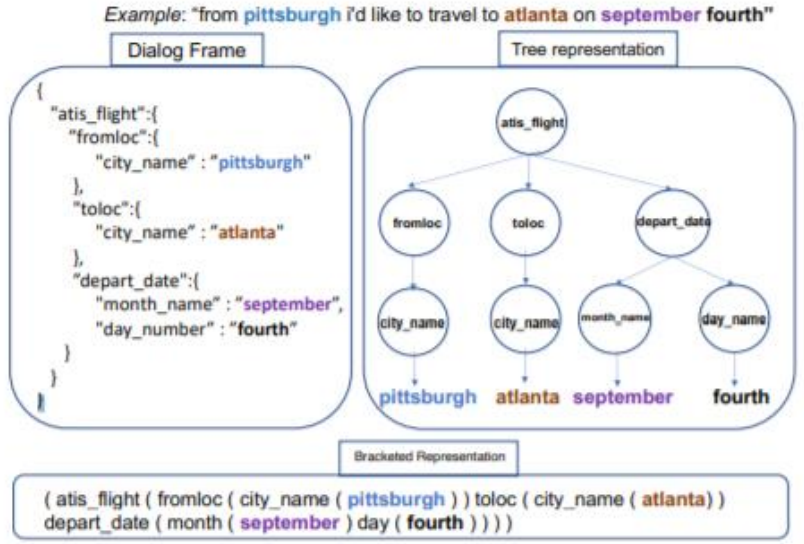

Figure 3: Representations proposed in this paper for an example from the ATIS dataset.

Method

- We learn to map a user's utterance $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$ to a a template-based tree representation. Our template-based generation is similar to sketch-based Seq2Tree decoding
- The translation is performed using four components: 1) an encoder 2)a slot decoder 3)a tree decoder 4)a pointer network

Method

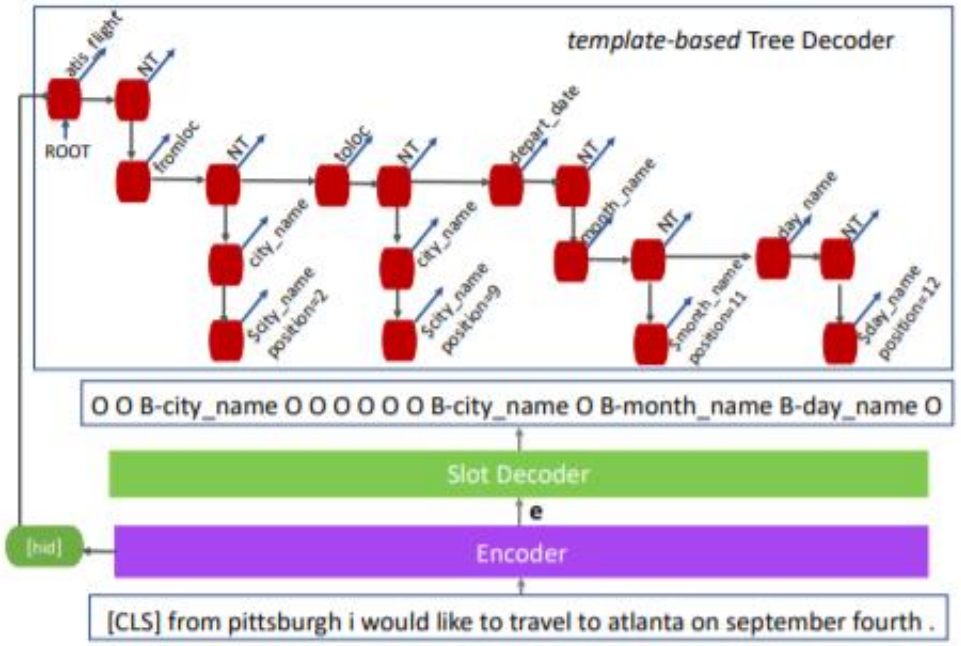

Figure 4: Proposed architecture.

Method—encoder,slot decoder

- We use BERT as the encoder to obtain token embeddings
- The slot decoder accepts embeddings from the encoder, which predicts the slot label for each token position $\hat{\boldsymbol{a}} = \widehat{a_1}\widehat{a_2} \ldots \widehat{a_n}$, the true slot label is $\boldsymbol{a} = a_1 a_2 \ldots a_n$. The decoder learns to predict Begin-InsideOutside (BIO) tags, since this allows the tree decoder to focus on producing a tree form.

$$\text{loss}_{SL} = -\frac{1}{n}\sum_{i=1}^{n} log\ \pi_{SL}(a_i|\hat{a}_{<i}, \mathbf{x})$$

Method—tree decoder

- The tree decoder works top down as shown in Figure 4 LSTM are used to generated symbols. In the example, the decoder generates atis flight NT, where NT symbol stands for a non-terminal.
- When a non-terminal is predicted, the subsequent symbol or token is predicted by applying the decoder to the hidden vector representation of the non-terminal.
- Each of the predicted NTs enter a queue and are expanded when popped from the queue. This process continues until no more NTs are left to expand.

$$\text{loss}_T = -\frac{1}{S}\sum_{s=1}^{S}\frac{1}{T_s}\sum_{t=1}^{T_s} log\ \pi_{TD}(z_t^s|z_{<t}^s, z^s, \mathbf{x})$$

Method—tree decoder

parent	children	Queue contents	Partially generated frame
head	ROOT NT_1	$[NT_1]$	ROOT ( )
NT_1	atis_flight NT_2	$[NT_2]$	ROOT (atis_flight ( ))
NT_2	fromloc NT_3 toloc NT_4 depart_date NT_5	$[NT_3, NT_4, NT_5]$	ROOT (atis_flight (fromloc ( ) toloc ( ) depart_date ( )))
NT_3	city_name NT_6	$[NT_4, NT_5, NT_6]$	ROOT (atis_flight (fromloc (city_name ( )) toloc ( ) depart_date ( )))
NT_4	city_name NT_7	$[NT_5, NT_6, NT_7]$	ROOT (atis_flight (fromloc (city_name ( )) toloc (city_name ( )) depart_date ( )))
NT_5	month_name NT_8 day_name NT_9	$[NT_6, NT_7, NT_8, NT_9]$	ROOT (atis_flight (fromloc (city_name ( )) toloc (city_name ( )) depart_date (month_name ( ) day_name ( ))))
NT_6	**$city_name**	$[NT_7, NT_8, NT_9]$	ROOT (atis_flight (fromloc (city_name (**$city_name**)) toloc (city_name ( )) depart_date (month_name ( ) day_name ( ))))
NT_7	**$city_name**	$[NT_8, NT_9]$	ROOT (atis_flight (fromloc (city_name (**$city_name**)) toloc (city_name (**$city_name**)) depart_date (month_name ( ) day_name ( ))))
NT_8	**$month_name**	$[NT_9]$	ROOT (atis_flight (fromloc (city_name (**$city_name**)) toloc (city_name (**$city_name**)) depart_date (month_name (**$month_name**) day_name ( ))))
NT_9	**$day_name**	$[\emptyset]$	ROOT (atis_flight (fromloc (city_name (**$city_name**)) toloc (city_name (**$city_name**)) depart_date (month_name (**$month_name**) day_name (**$day_name**))))

Table 1: Actions taken to generate the frame representation of the sentence, *from pittsburgh i'd like to travel to atlanta on september fourth.* "NT" refers to non-terminals.

Method—Pointer network

- We predict positions for every terminal, pointing to a specific token in the user's utterance.
- The pointet network loss $loss_{P_T}$, is the categorical cross entropy loss between pt and the true positions.

$$p_t = \arg\max_i \ \mathbf{softmax}(g(h(z_t^s) \odot e(x_i)))$$

- The total loss is calculated by:

$$\text{loss}_{-G} = \text{loss}_{SL} + \text{loss}_T + \text{loss}_{PT}$$

Method-global context

- We found that the tree decoder tends to repeat nodes, since representations may remain similar from parent to child.
- We overcome this by providing global supervision. This global supervision does not consider the order of nodes, but rather rewards predictions if a specific node is present or not in the final tree.Let $z_1, z_2, \ldots, z_K$ is the unique set of nodes present in the reference tree and $N(z_k)$ is the time z_k occurs.

$$\text{loss}_G = -\sum_{k=1}^{K} \frac{N(z_k)}{\sum_j N(z_j)} log\ \pi_G(z_k|\mathbf{x}) \qquad \text{loss}_{\text{weighted G}} = \text{loss}_{-G} + \text{loss}_G$$

Experiments

Model	ATIS				Simulated			
	gen-acc	spec-acc	gen-f1	spec-f1	gen-acc	spec-acc	gen-f1	spec-f1
Proposed method,-G	61.74	59.53	88.50	87.29	91.48	90.75	99.64	98.63
Proposed method,+unweighted G	62.21	60.23	87.33	86.81	91.85	90.68	**99.97**	98.63
Proposed method,+weighted G	**72.00**	**70.54**	**89.32**	**88.87**	**92.14**	**91.12**	**99.97**	**98.76**

Table 2: *+/-G:* with or without the global context loss function. *gen:* generalized form metrics and *spec:*results with the specific form. *acc:*accuracy and *f1:* f1-score on parent child relationships.